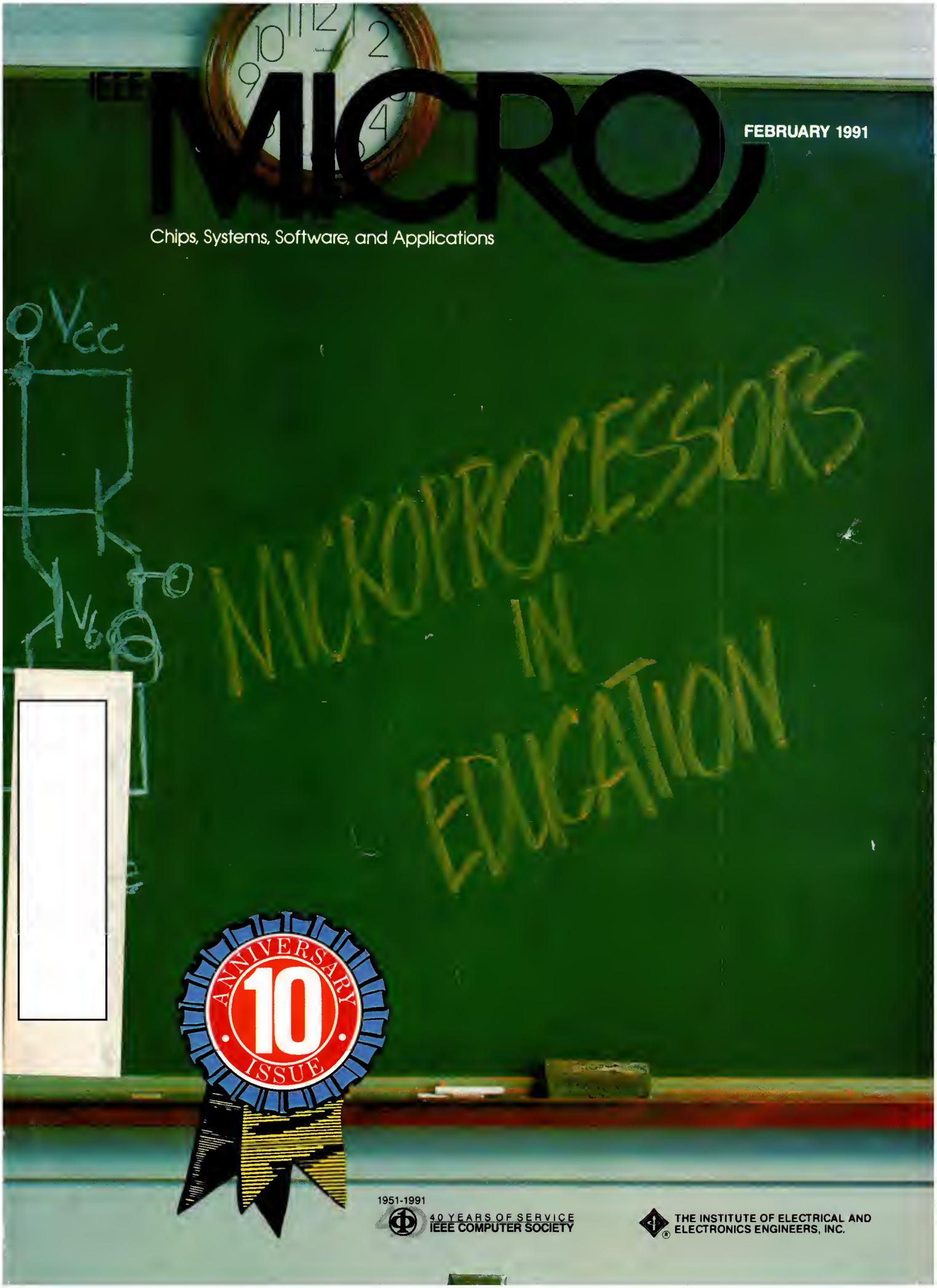

IEEE
MICRO
FEBRUARY 1991
Chips, Systems, Software, and Applications
MICROPROCESSORS IN EDUCATION
ANNIVERSARY
10
ISSUE
1951-1991
40 YEARS OF SERVICE
IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

10th Anniversary

Celebrating IEEE Micro's

February 1991

FEATURES

Cover by Design & Direction

Circulation: *IEEE Micro* (ISSN 0272-1732) is published bimonthly by the IEEE Computer Society, PO Box 3014, Los Alamitos, CA 90720-1264; IEEE Computer Society Headquarters, 1730 Massachusetts Ave., NW, Washington, DC 20036-1903; IEEE Headquarters, 345 East 47th St., New York, NY 10017. Annual subscription: $21 in addition to IEEE Computer Society or any other IEEE society member dues; $38 for members of other technical organizations. This journal is also available in microfiche form.
Postmaster: Send address changes and undelivered copies to *IEEE Micro*, PO Box 3014, Los Alamitos, CA 90720-1264. Second-class postage is paid at New York, NY, and at additional mailing offices.

IEEE MICRO

Published by the IEEE Computer Society

DEPARTMENTS

IEEE Computer Society
PO Box 3014, Los Alamitos, CA 90720-1264
(714) 821-8380

* Submit six copies of all articles and special-issue proposals to Dante Del Corso, Dipartimento di Elettronica, Politecnico di Torino, C.so Duca degli Abruzzi, 24, 10129 Torino, Italy; phone +39-11-556-7244; Compmail: d.delcorso; Internet: delcorso@pol88B.to.cnr.it; Bitnet: delcorso@itopoli

From the Editor-in-Chief

Launching the second decade

When first I heard I was being proposed as the next editor-in-chief of *IEEE Micro*, I immediately turned to scan past issues, from the premiere issue to the most recent one. (I am proud to own a complete collection of this magazine.) Besides giving me an overview of the evolution of electronic technology in the last 10 years, these issues helped me to really understand that being the editor-in-chief of *IEEE Micro* is both a big honor and a big challenge.

During the tenures of previous EICs, the quality of this magazine continued to rise. *Micro* is now a magazine with a truly international attitude (the most pronounced in the IEEE landscape, I think). It publishes regularly timed topic issues and has established a tradition of good tutorials and interesting departments.

My next thought centered on Richard Jaeger, Peter Rony, James Farrell, and Joe Hootman, the previous EICs. Should I thank them for delivering a magazine in such good shape or blame them for making the EIC job more demanding? Well, I like challenges (to a given extent, please do not take me literally on those last three words!). So, Joe, Jim, Peter, and Dick, as a reader I thank you all for what you did for and with *Micro*; I begin to understand the complexity of your task.

As EIC, I extend this thank-you list to include the authors, Editorial Board members, and the Los Alamitos staff; it is a true privilege being selected to work with such a team.

My first task is to ensure that the reputation of the magazine remains intact and, if possible, to improve on it. To translate these good intentions into printed pages, I will need a lot of help from the Editorial Board members, authors, editorial staff, and readers too. One of the reasons that convinced me to take on this task was the strong cooperation I received in my past experiences as co-guest editor of the special European issues.

On those occasions I realized the quality of a magazine of this type rests on three pillars:

- the authors and Editorial Board members, who assume responsibility for technical quality (all papers go through a peer-review process);
- the editorial staff, who guarantees that information is delivered in the most understandable and effective way; and
- the readers' feedback, which can influence topic selection and magazine content.

My analog background resurfaces here: I think of *Micro* as an operational amplifier, with "gain"

0272-1732/91/0200-0002$1.00 © 1991 IEEE

provided by authors and staff, and a transfer function defined by a "feedback network" made up of the Editorial Board and readers.

Then what is left for the EIC to do? Besides providing part of the "gain" and "feedback" just mentioned, I think a main task of the EIC is to collect new ideas and stimulate cooperative work within the Editorial Board and among authors, staff, and readers. To make this cooperation effective, please freely offer any contribution, suggestion, proposal, or criticism. They make the basis for a successful magazine.

I have been told that this is the first time that the Computer Society has experimented with an EIC residing outside the United States. This experiment is made possible by today's information technology, especially electronic mail networks, fax machines, and the intercontinental telephone links. *Micro* will play an active role not only in disseminating technical information but also in testing complex electronic systems.

Let me again enforce the concept that mainly authors and readers make up this magazine; your contributions enrich us all. If you have a paper ready, send it to me. If not, start now to prepare one or two. If you know of others who have papers ready, encourage them to contact me. (An author's information sheet is available from the CS Publication Office in Los Alamitos, California, at the address listed on p. 1 of this issue.)

This issue on education not only marks the 10th anniversary of *IEEE Micro* but also forms a good link between the first 10 years and the next decade, since it addresses a key point for developing the future. First, we look back to 1981 with a short article written by publisher True Seaborn and the former EICs. They recall the formation days of *IEEE Micro*, its original charter, and their experiences in continuing the magazine's founding ideals. See also Hubert Kirrmann's Micro World, which takes us back to the personal computer state of the art in 1981.

Microelectronics people in these 10 years moved from 16-bit, 8-MHz, and less than 100,000-transistor devices (the MC68000, on the cover of the first *Micro* issue) to DSPs and microprocessors with on-board FPU, cache memory, more than 1.2M transistors, and 40-MHz clock cycles (various articles in 1990 *Micro* issues). The key question is whether such performance improvement will continue at the same rate. Take a closer look at what we can expect in the future in the article by Ware Myers, "The Drive to the Year 2000," and Michael Slater's column on evolving architectures. Myers forecasts the landscape for electronic systems in the years to come, providing figures based on reliable technology forecasts.

Communications are a key issue in the development of high-performance architectures and applications.—A.T., Mashhad, Iran

In the mailbag

June 1990

I liked ...

Design simulators.—H.P., Mashhad, Iran

Slater's comments on RISCs ... in Micro View. I would add that the best name is PISC for pipelined instruction-set computer.—R.S., Tucson, AZ [Anyone agree with this proposal?—D.D.C.]

"The 68040 Processor" ... well written and easy to follow. What is RISC? ... Good!—V.R.N., Punta Arenas, Chile

"The 68040 Processor" .... I missed Part 1.—R.K.C., Ahmedabad, India [A copy is on its way to you.—D.D.C.]

All information about microcomputers ... good articles. You must continue so.—V.B.H., Valparaiso, Chile

I would like to see ...

More pipeline articles ... multiple execution units and instruction scheduling at the microprocessor level.—R.S., Tucson, AZ

More on 68000 and 88000 families.—R.K.C., Ahmedabad, India

More about computer's internal frame in the microprocessor articles.—V.B.H., Valparaiso, Chile

August 1990

I liked ...

Emulators and parallel computer

I would like to see ...

Applications of micro[processor]s in real-time simulation of flight.—T.P.H., Singapore [Anyone want to take up the challenge?—D.D.C.]

October 1990

I liked ...

Micro Law.—R.J.H., Houston TX

I would like to see ...

More details (performance comparison) about floating-point DSP chips ....—P.T., Pittsburgh, PA [A special DSP issue is coming; I will send your proposal to the Guest Editor.—D.D.C.]

General

Would like to take a closer look at fixed-point, 16-bit processor without MMU or cache. [Some past issues of *Micro* described these processors.—D.D.C.]

TI's 320C50 processor ... features several unique (to my knowledge) capabilities ... implements a boundary scan testing with JTAG interface .... Probably somebody ... could write a few pages for *Micro* readers?—J.R.C., Warsaw, Poland

processing systems. Simon Peyton Jones describes in "A Futurebus Interface from Off-the-Shelf Parts" one way to overcome this bottleneck. He uses an advanced asynchronous protocol, which can squeeze ultimate speed from available bus interface technology.

The amount of things we must know to comply with the increasing complexity of information technology and microelectronics expands every year at an overwhelming rate. As a teacher I understand that we cannot just add lots of courses, textbooks, and lab work to students' curricula; our schools and universities need better organization of the education process. The remaining articles in this issue discuss the most efficient way to bring advanced microcomputer technology to students, so that they may follow—or, even better, lead—tomorrow's developments.

One article, "Dedicated Tools for Microprocessor Education" by Jean-Daniel Nicoud, presents hardware and software teaching aids developed over several years. Another major article, "Fun and Games and Microcomputer Interfacing" by John Fulcher, tells how a course on computer interfacing can be made more interesting or even enjoyable. Students, as a result, achieve the maximum involvement and benefits.

Then a set of shorter articles describes specific tools and experiences in teaching in the microcomputer field. These articles address hardware tools (by L. Howard Pollard and Ramiro Jordan), software environments (David Russell and Kirtley Haden), a mix of both (Thomas Schultz), and course organization (Douglas Hall). Former Editorial Board member Marlin Mickle proposed the majority of these articles from papers presented at last year's Modeling and Simulation Conference.

As a whole, these articles provide an overview of tools and techniques that can effectively support the education process in this area. The education methodologies arena is still open; and we leave evaluation and comparison of these proposals to you readers. I invite you to consider them only a starting point to stimulate a debate on the best way to improve the yield of education in microelectronics and computers.

Looking back

IEEE Micro is 10 years old! Since 1981, *Micro* has fulfilled its charter to provide readers with technical information that will help them keep abreast of developments in the constantly evolving microcomputer industry.

It's been an exciting time for everyone involved, one we felt readers might be interested in reviewing. We asked Publisher True Seaborn and each past editor-in-chief to recall some of the highlights of our first decade.

True Seaborn

Peter R. Rony

Richard C. Jaeger

James J. Farrell, III

Joe Hootman

Sometimes a good idea beats a good plan (but not often!)

True Seaborn

IEEE Micro was one of those bursts of sheer exuberance we don't often see these days in the Computer Society. Good thing, too. Not that *Micro* wasn't a good idea. But today we try to invest in a little more planning before we launch a new magazine.

Let me set the scene for you. It was February 29, 1980, at a Board of Governors meeting in San Francisco. Compcon Spring 80 had ended the previous day, and the board was holding its first meeting of the year. It had just approved the launching of *IEEE Computer Graphics and Applications*, a new quarterly to appear in 1981, and one we hoped would catch the wave of enthusiasm we'd seen in such trends as the rapid growth of Siggraph and the appearance of the National Computer Graphics Association. We had conducted a membership survey, aimed specifically at gauging member interest in topics for potential new magazines, and computer graphics had emerged as the clear winner. We were confident we'd done our homework. The *CG&A* motion passed unanimously, and I left the room to report the news to Marilyn Potes and Joe Schallan back at the Publications Office in Long Beach.

I couldn't have been gone for much more than five minutes. When I returned, one of the board members—I can't remember who—turned to me and said, "Well, we now have not one new magazine but *two*!" I was glad I hadn't stayed out of the room any longer than I had.

Somehow, during that brief five minutes or so, board member Bob Stewart had introduced the notion of a second new quarterly to appear in 1981, gotten it seconded, and passed. I'm told that Bob based his arguments on the need to provide practical, detailed information on

0272-1732/91/0200-0005$01.00 © 1991 IEEE

microprocessor design and application for the bench engineer. I never was very clear on what Bob had meant exactly by "bench engineer," but obviously whatever he said was persuasive. And thus *IEEE Micro* was born.

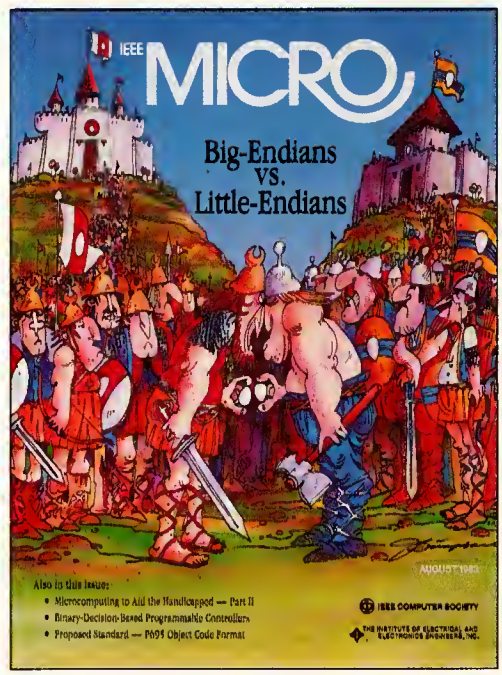
Dick Jaeger of Auburn and Peter Rony of VPI&SU were selected as editor-in-chief and associate editor-in-chief. Their joint message in the inaugural issue (reproduced here in the accompanying box) presented the technical scope and intended audience of the magazine.

Micro hasn't deviated much from the blueprint that Dick and Peter laid out 10 years ago. The basic approach of providing detailed technical information on microprocessor design and application, aimed at computer professionals including both designers and users, is still one we follow. Jim Farrell of VLSI Technology, Inc., who succeeded Peter Rony as EIC in 1985, increased our internationalism, with special issues organized each year by European and Asian members. Joe Hootman of the University of North Dakota, who succeeded Jim, has maintained that tradition. And, of course, with the appointment of Dante Del Corso of the Politecnico di Torino as our new EIC—the society's first from outside the US—that international thrust has been made even more apparent.

Technical scope and audience

Our audience will include professionals in the design and use of microprocessors and microcomputers, from chips through systems. Hardware configuration, interprocessor communication, analog signal interfacing, software design, numerical software, and process control are just a few of examples of topics relevant to us. Authors should strive for readability in their papers and should keep in mind that people with a broad range of backgrounds will be reading the articles, not just electrical and computer engineers. We encourage authors to use as many clear, explanatory figures as possible—they can make a substantial contribution to the readability and vitality of a technical article. Material explaining concepts or introducing terminology should be included, perhaps separated from the main text, whenever such material will enhance the reader's understanding.

Tutorial articles on a wide range of subjects will also play a critical role in the success of this magazine. Subjects such as digital filtering, fast Fourier transforms, and numerical software are appropriate for tutorial articles. However, papers of the "this-is-what-we-have-done-and-isn't-it-wonderful" variety, which simply describe a microprocessor application but are devoid of technical detail, are not appropriate.—Jaeger and Rony



Bob Stewart's instincts proved right. There was a sizeable appetite for detailed information on microprocessors within our own membership, and *Micro*'s early circulation growth reflected that interest. By the end of 1981 we had reached 16,700 total paid circulation, and we continued to grow at an annual rate of 18-19 percent for the next two years. We peaked at 23,450 in 1983, and we've been holding fairly steady at about 19,000 for the last three years.

Our first issue appeared in February 1981, with a picture of the Motorola 68000 on the cover counterbalancing the lead article by Robert Noyce and Marcian Hoff, Jr., of Intel. The cover blurb inside says the 68000 "typifies the state of the art in microprocessor design and fabrication," although the lead article, a history of microprocessor development at Intel, mentions the introduction of the iAPX432, the first 32-bit microprocessor.

By some standards 10 years isn't a very long time. But in the computer business—and in the publishing business—it's a very long time indeed. Certainly our way of producing this magazine has changed dramatically over the past decade. In 1981 we were still handling manuscripts in hard copy, and editors and layout artists plied their crafts using blue pencils, Xacto blades, and light tables. Manuscripts were rekeyed, proofed, corrected, reproofed, recorrected, etc. Production cycles were periodic tests of endurance and sanity. Fixed costs/editorial page for *Micro* were in the neighborhood of $100.

Today we receive our manuscripts on disk or via modem. Editors edit on line and negotiate changes with authors via

electronic mail or fax. We use Pagemaker for layouts, and we ship disks to the printer for direct output to film, including embedded line art. *Micro*'s fixed costs/editorial page have dropped by 35 percent. Production cycles are still stressful, and they still usually require more than 40 hours per week (this is, after all, still the publishing business). But life at the Publications Office today bears little resemblance to life here a decade ago.

It has been a privilege for me to work with the fine professionals, volunteer and staff, that have made this magazine what it is today. On this occasion of *Micro*'s 10th anniversary, I'd like to take this opportunity to salute the EICs and managing editors who have worked so hard during these past 10 years to make it a success:

Editors-in-Chief
Richard C. Jaeger, 1980-1982
Peter R. Rony, 1980-1985
James J. Farrell, III, 1985-1988
Joe Hootman, 1989-1990
Dante Del Corso, 1991-Present

Managing editors
Joe Schallan, 1980-1985
Richard Landry, 1985
Marie English, 1985-Present

A magazine is born

Peter R. Rony and Richard C. Jaeger

The year 1980 was special for the CS. That year the volunteers, after considerable debate, boldly broke new ground and committed the CS to two new publications. Both magazines carried a different mission of service to CS members than did the CS *Transactions*. Four particularly appropriate comments from our records (see the box on p. 61) will give you a flavor of the 1979-80 debate.

The creation of these magazines required a shared vision by a number of volunteers. We acknowledge and honor Richard Merwin, Robert Stewart, Rex Rice, Roy Russo, Portia Isaacson, Tse-Yun Feng, T.H. Bonn, Ed Parrish, David Stomberg, True Seaborn, Oscar Garcia, Dick Simmons, Mike Wozny, Harry Hayman, Martha Sloan, Paul Hazan, H. Fleisher, and others—collectively, *Micro*'s founding fathers. They can all be proud of the magazine they created.

Ten years later, we take pleasure in recalling the early hopes, debates, concerns, and objectives. Were they fulfilled? We believe so.

The creation. Once the decision to create *Micro* was made, the pace of decisions was breathtaking. Though the magazine's title and scope was not set until April 1980, the first issue arrived for editing at the Long Beach office in October 1980. Looking back now, we find it amazing that the pieces came together so quickly, a tribute to the positive attitudes of all individuals associated with the "birth" of the magazine.

The implementation. The Board of Governors, in its desire to serve a larger number of CS members, set the basic style of the magazine. The published materials had to be readable and of use to the hypothetical bench engineer who designed hardware or software. Both of us, during our terms as EIC, tried our best to implement this vision.

Micro became a reader's magazine. The collective responsibility of authors, reviewers, and editors provides clear, timely, informative, and compelling articles to CS members. We always perceived the willingness of an author to contribute a manuscript to *Micro* as being something special. Editors and reviewers tried to assist the author in producing a manuscript that was not only approved for publication but also was enjoyed by readers when published. We tried our best to make our writers feel comfortable with the editorial process.

The wonderful working relationship we had with the editorial staff, True Seaborn (editor), Joe Schallan (managing editor), Ware Myers (contributing editor), Jay Simpson (art director), and many others, was an essential feature for the success of the magazine.

To novices as technical editors, the editorial production team was the best news to come our way once we woke up in the morning to the sobering knowledge of our numerous responsibilities. We gathered the material to be included in the magazine. Once we mailed it to the Publications Office, the staff took over and produced the magazine. They coordinated the innumerable details that go into a magazine which can, in good conscience, be mailed to readers. Mercifully, most of the production process remained transparent to us, the mark of a skilled editorial production team. Only now, with the appearance of PC-based desktop publishing software, can we begin to appreciate what they did.

What a pleasure it was to work with Joe Schallan. No more important person exists in the life of an EIC than the managing editor, who has a role much like the fairy godmother in Cinderella. With a flick of the wand (current version: word processor), a pile of heavily edited, dog-eared, motley looking manuscripts becomes a professional magazine on glossy paper.

Our interaction with Joe was so important that we followed him to his "electronic cottage" in Arizona, where he continued to serve as managing editor for several years. Out of the magazine's editorial budget, we provided him with a Zenith Z100 personal computer and tried to make editorial life in the Arizona boondocks as comfortable as possible.

We also have a special debt to Jay Simpson, art director par excellence, who has consistently produced the very best magazine covers in the computer business, with only *Byte* magazine even being in the competition. How about the Big-Endians vs. Little-Endians (August 1983) and the Standards (August 1984) covers you see reproduced here? They are classics. Outstanding cover art is a measure of the CS's pride in its magazines, a pride exhibited with every issue. We are very lucky to have Jay working with us.

Joe Hootman, in his retrospective, comments on the significant role that *Micro*'s associate editors, and the skilled reviewers whom they select, play in the identification and improvement of technical articles. We concur.

The most sensitive time in the birth of the new magazine is the first several issues, which must be very special to capture the imagination of readers. We gratefully acknowledge our editorial colleagues who were there at the beginning: Andrew Allison, Tuvia Apelewicz, Walter Beam, J. Thomas Cain, Alvin Despain, Patrick Fasang, Donald Feinberg, John Hennessy, Gerald Kane, Fred Liquori, Richard Markowitz, Jean-Daniel Nicoud, and Deene Ogden.

Tom Cain expanded his extensive involvement with the magazine through his appointment as Associate EIC for Peter. Ask any EIC how important is a close, working relationship with the Associate EIC, and you will understand how we both feel about Tom's participation in the editorial process.

Robert Stewart ultimately became an Associate Editor with specific responsibility for the Micro Standards column. We considered Bob to be a very special member of our team. Bob continually kept us on track about the importance of readability and applicability of articles to the bench engineer. Bob's retrospective on the early years of *Micro* appeared in the August 1986 issue.

Soon after the magazine was created, we initiated the search for both appropriate and also unique features that would complement the technical articles and make *Micro* somewhat special to readers.

The principle special features during our tenures in office were Bob Stewart's Micro Standards, Richard Stern's Micro Law, Victor Nelson's New Products and Product Summary, as well as Micro View and Access. Three of these, Micro Standards, Micro Law, and Micro View represented a departure from other CS publications. Today, our early conviction that standards, law, and politics were fundamental elements in the progress of the microcomputer industry can be viewed as being on target.

Members of the United States Congress contributed periodically to the early *Micro* through the Micro View special feature. Senator Charles Matthias contributed in August 1983, Representative Don Edwards in December 1983, Representative Ed Zschau in February 1984, Senator John Chafee in February 1985, and Senator Patrick Leahy in April 1985. We regarded the opportunity for points of view, on issues of mutual interest, to be expressed by our elected representatives in the pages of *Micro* to be both important and appropriate. The door should always be open to such individuals to contribute their concerns to CS publications.

Being EICs of *IEEE Micro* was an honor and pleasure, but hard work. We thank each person who participated in the process.

A decade of contributors

James J. Farrell, III

When I was asked to write for the 10th anniversary issue of *Micro*, my first thought was of all the in-depth technical articles and departments that it has brought readers over the years. By my calculation, *Micro* has published well over 300 refereed articles in the past decade. That represents an enormous body of work on the part of authors, volunteer editors, referees, and CS staff. If we were to thank all who have contributed to *Micro* over the decade, it would take many pages. I would like to share some recollections of just a few of the *Micro* contributors that I was privileged to work with during this decade.

My first contact with *IEEE Micro* came at Compcon 80, at the Cathedral Hill Hotel in San Francisco, on February 29.

I have since forgotten the topic of my conference paper, but after my presentation, I ran into Pat Fasang, who I knew from other technical conferences and seminars. Pat told me that he had just joined the Editorial Board of a new CS magazine called *IEEE Micro*. Since microprocessors were my main professional area of interest, I discussed this new magazine further with Pat. At the end of our conversation, Pat commented that he was seeking photographic candidates for the cover of the premier issue. He asked if I could provide an interesting, color photomicrograph of the Motorola MC68000 microprocessor chip. I forwarded the request through the proper channels at my employer, Motorola. A few months later, when the premier issue arrived, I found the MC68000 chip photomicrograph was indeed selected for the cover.

I first met Tom Cain in person at a Motorola pre-Wescon press meeting held in Phoenix in November of 1983. (He had a lot less gray hair then.) At that time, Tom was associate EIC of *Micro*. During the previous two years, I had contributed personally to *Micro*, and had assisted others at Motorola contribute technical articles. Some months earlier, I had been appointed to the Editorial Board of *Micro*, but this was the first occasion that I had actually met in person with one of my "bosses."

Tom is an academic, and I think this was the first time he had ever been to a commercial "trade press" meeting. We both learned a lot. Tom was certainly a significant contributor to *Micro* during its "early days." He remains extremely active in various IEEE and CS positions outside of publications, but I can usually count on him to show up at a CS Publications Board meeting to see what's cooking.

Peter Rony worked very hard for *Micro* during its first four years, and appointed many of the second-generation Editorial Board members, including me. Peter initiated an effort to get the most current technology possible for readers, and then to "fast-track" the refereeing of time-sensitive articles. Even though Peter teaches and holds a PhD in chemical engineering, he is extremely knowledgeable on the subject of microprocessors and computers.

Doug MacGregor was a microprocessor designer and a coworker of mine in Austin, Texas. He is one of the best technical authors that I know. In addition to two *Micro* Article of the Year awards, in 1983 and 1984, Doug was one of the recipients of the IEEE Browder-Thompson Award for outstanding authors in 1983. After his awards, Doug completed his PhD work at the University of Kyoto, and went on to become a founder and current chief executive officer of Solbourne Computer, Inc. I occasionally run into Doug at technical exhibits. He is a real-life computer-industry success story.

Yousif Imam is a research scientist with IBM who specializes in speech recognition. His excellent 1986 article for *Micro* on speech recognition was the cover feature, and was awarded the *Micro* Article of the Year for 1986. He and his associate followed this 1986 publication with their further research, which was published in the August 1990 issue. His work on this latest article was barely completed in time. A few days after the August issue of *Micro* went to press, his homeland was invaded. Very little information can get in or out. Since mail delivery is unlikely, he has probably not yet seen his article in print. I have not met him personally, but I certainly would like to do so.

The role of the EIC

Joe Hootman

The EIC of any CS publication has a basic responsibility to its readers. Readers have every right to expect quality articles that are written in an understandable style and contain up-to-date information in each issue.

Upon assuming the EIC responsibilities for *Micro*, I soon found that the time that could be devoted to being EIC was way beyond what I had anticipated. I spent two to three hours each day and devoted substantial time on the weekends to these responsibilities. To help get the job done, and involve it directly in the editorial content of the magazine, I relied heavily on the Editorial Board.

The EIC must surround himself with people who are willing to put the reader 's interests ahead of their own self interests. Thanks to the volunteer nature of the CS, the problems of self interest are reduced or eliminated altogether. Since *Micro* 's Board was also a reasonable mixture of both academic and industrial members, I was able to select the member most likely to provide a fair and impartial review of a particular paper. This review process is extremely important to the reader and to the author. It ensures the reader that the material is up to date, accurate, and readable. It ensures the author that when the finished paper is published it will be a high-quality piece that meets the IEEE standard for publishing and reviewing and warrants some pride. I have found that authors and academic administrators use *Micro* as a part of their basis for promotion and tenure.

As EIC, I felt responsible for every article in each issue. I read in great detail each paper that was eventually published. I also read all of the columns that were published when possible.

It would not be fair to discuss the role of the EIC without discussing the CS publishing operation. I think it would

continued on p. 61

The Drive to the Year 2000

The microcomputer industry faces a number of challenges in the coming decade. As the performance of the technologies grows exponentially, so will the costs of developing and fabricating them. Since such funds will ultimately come from the marketplace, it must grow accordingly to support this development. The resulting challenge, therefore, is to expand the microcomputer market through new applications and more intensive consumer use of existing products.

Ware Myers

Contributing Editor

Popular forecasters of the future tend to believe that anything they imagine can be implemented shortly thereafter. Part of the blame for this expansive state of mind rests with the computer community. Indeed we have accomplished many wonders in the past 40 years, and it appears additional wonders are *technically* feasible by the year 2000.

During these four decades, however, we also researched tunnel diodes, perceptrons, wire memory, and many other technologies that somehow got sidetracked. One of my early articles for *Computer* magazine (in 1977) reflected the then bright glow of magnetic bubble memory. A couple of years later I basked briefly in the early morning warmth of optical storage, a technology still struggling for a foothold in the computer market. Another example: The original version of the Next computer employed an optical disk, but the second model retreated to magnetic disk.

So, the future is out there, but it is obscured by a great deal of fog. Well, let us press on.

Some technologies have proved to be surprisingly successful. The semiconductor march promises to continue for another decade. Intel chairman Gordon Moore said as much in an IEEE Computer Society Compcon keynote address in 1989. Consequently, all those facets of information technology that rest in part on semiconductor technology—processors, memory, digital signal processing, and communications—should continue to drop in price and increase in performance.

The progress of magnetic recording has not been prompted by a well-known "law," but it has been equally remarkable. Other technologies have also advanced rapidly and new technologies continue to appear. This technological progress will result in an enormous expansion in computing capability. More capability can support more ambitious applications, but these have to be found.

Still, we must remember that semiconductor and magnetic-recording technology are the exceptions among the technologies. Both have advanced rapidly for a long time. Some technologies contributing to information products are not so lucky. Some have advanced slowly—or fitfully. Others have lost out because they did not manage to catch up with existing technologies.

The laws of the semiconductor

The advance of semiconductor technology has been so consistent for so many decades that industry seers concocted "laws" to characterize past progress and to project the future. Here is a sampling of laws from Moore:

- The number of transistors per chip doubles every two years (a law originally handed down in 1975 and still going strong).
- Feature size has been cut in half every six years.

0272-1732/91/0200-0010$01.00 © 1991 IEEE

Table 1. Projected schedule of increases in memory density per chip.

Memory density (Mbits)	Feature size (µm)	Type radiation	Commercially available
4	0.80	Visible light	1990
16	0.50	Visible light	1992
64	0.33	Electron beam or ultraviolet	1995
256	0.20	X ray	2000
1,024	0.10	X ray	?

- The amount of charge needed to distinguish between two states has been reduced from five million to 100,000 electrons—a factor of two each generation.

An 11-member expert panel offers a somewhat different version:

> **Logic and memory component density is expected to increase by a factor of 10 every five years through the end of the century.[1]**

(The panel was established by the Aeronautics and Space Engineering Board of the National Research Council to project the information and computer technologies that would be possible by 2000 to support aeronautics if sufficient resources were available.)

Now, in the beginning of the last decade of the second millennium, the effect of these and other laws have brought us to 4-Mbit memory chips and one-million-transistor microprocessor chips. If the laws continue to hold, we should reach 256-Mbit memory chips and 50-million-transistor microprocessor chips by 2000. Researchers do not expect to reach the ultimate limits set by the laws of physics during this decade.

Memory. Manufacturers began producing the 4-Mbit chip in large volume last fall.

> **Quadrupling the capacity of each generation of DRAMs [dynamic RAMs] has been achieved by reducing cell size to one third and increasing chip area by one half.—Fujio Masuoka**

The next generation, 16-Mbit, is being developed by many manufacturers. Hitachi researchers have announced the fabrication of the following generation—64 Mbit—in the laboratory, but yields are still extremely low. The researchers are using electron-beam technology with a feature size of about 0.33 µm. Volume production of the 64-Mbit chips is said to be about five years off. In the meantime, research is in progress on a new way of applying light waves called phase shifting. This method may extend optical lithography down to the 0.33-µm level, or even lower. With X-ray lithography, researchers can reduce feature size to 0.20 micrometers, making the 256-Mbit chip possible, perhaps by 2000. Table 1 summarizes this density progression.

Problems, such as supply voltage and cell capacitance, turn up all along this projected trail, though researchers expect to overcome them. Masuoka[2] most recently surveyed these problems.

Logic chips. In microprocessors, Intel's 80486 and Motorola's 68040—each with more than one million transistors—represent the current top of the line. Intel plans to introduce the 80586 in 1992, according to executive vice president Craig Barrett. One chip will hold four to five million transistors.

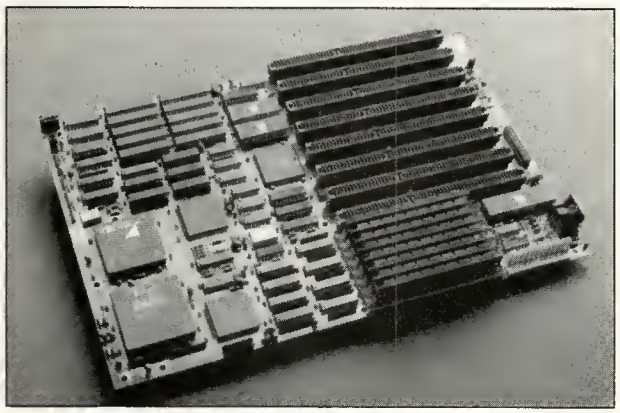


Figure 1. Pioneer Computer's 80486-based motherboard.

(Figure 1 features an 80486-based motherboard by Pioneer Computer that demonstrates the current state of the personal computer art. In addition to its microprocessor and eight slots for optional boards, it can house up to 256 Kbytes of secondary cache memory, up to 32 Mbytes of main memory, and a math coprocessor.)

Bruce Patterson, Intel's managing director in Australia, speaking at a conference in Sydney last June, looked still farther out to the 22-million-transistor 686 by 1996 and the 100-million-transistor 786 by 2000. The expansion of microprocessor capability is outlined in Table 2 on the following page.

> **Over the last 30 years, the MIPS/dollar ratio of computers has increased by a factor of over one million.—Patrick P. Gelsinger, Paolo A. Gargini, Gerhard H. Parker, and Albert Y.C. Yu[3]**

By 2000, Gelsinger and his group at Intel project 50-million transistors on a one-inch-square die operating at 250 MHz.

Table 2. Projected growth of microprocessor power.

Designation	No. of transistors (millions)	Commercially available	Clock Rate (MHz)
486	1.2	1990	40
586	4.5	1992	60
686	22.0	1996	100
786	50-100.0	2000	250

They assume four parallel processors on this chip, operating at 750 MIPS (million instructions per second) each. In addition, transistors would be available for floating-point operation, data and instruction caches, vector units, a graphics unit, and various controllers.

Moreover, an interface better suited to human characteristics, involving speech synthesis, voice recognition, motion video, and image recognition could be accommodated on this size of transistor budget. This budget could also accommodate fault-tolerant design, redundancy, built-in self-test, and error detection and correction.

Wafer-scale integration. The one-inch-square chip that Intel anticipates is about four times the area of today's largest chips. In the meantime, research continues in the US, Japan, and Europe on advancing to wafer-scale products. In Japan, progress is ahead of schedule, Tadashi Sasaki reported to the 1989 International Conference on Wafer-Scale Integration. He expects to see this technology in use before 2000.

"This much is certain," Sasaki added. "New process technology oriented toward wafer-scale integration will be introduced gradually into chip design, as designers find new ways to resolve the problems."

Digital signal processing. A DSP is basically a special-purpose processor designed for exceptionally fast execution of certain instructions (particularly multiplication and addition). The multiply-accumulate combination occurs in many algorithms employed in processing real-time signals. Thus, processing signals coming in from the analog world via an analog-to-digital converter demand this kind of computing power. Similar digital processing has to be performed before the results are sent out via a digital-to-analog converter to the real world.

> **Between 1976 and 1986 a three-order-of-magnitude reduction in cost, size, weight, and power consumption occurred.—L. Robert Morris[4]**

> **Between 1986 and 1988 the state of the art advanced from fixed-point to floating-point architectures, allowing an increase in dynamic range and precision.—Stephen A. Dyer and L. Robert Morris[5]**

Since the DSP is essentially a type of microprocessor on a chip, it will benefit during the 1990s from increasing density. A likely development is that the DSP function will migrate to the same chip that holds the other functions necessary to the operation of a complete computer system. The signal processing involved in such activities as speech recognition, speech synthesis, three-dimensional moving graphics, image and video processing, and data compression and reconstitution will no doubt occur on this one multifunctional chip. With the data remaining on this chip during several kinds of processing, speed can be greatly increased.

In the past, it has been impractical to integrate both the analog circuitry involved in going from the analog realm to the digital realm (and vice versa) and digital processing circuitry. Digital switching "pumps" the chip's ground level to a degree that interferes with the precision of the A/D or D/A conversion processes. Recently, however, a new conversion technology moves the operations requiring precision into the digital realm. This technology promises to make integration of A/D and D/A conversion feasible on this ever-expanding chip.[6]

Edward A. Lee[7] believes that DSPs may become the key to the much touted—but little realized—integration of telecommunications and computation.

Communications chips. Caught up in the excitement of the greatly increased bandwidth provided by fiber optics, we sometimes lose sight of the fact that, from another vantage point, a communications network may be thought of as a vast collection of computers connected by transmission links. Those computers, or switches as the telephone companies call them, are also based on semiconductor technology. Since the fiber optic links will have enormous transmission capacity, they will be capable of carrying telephone voice, teleconferencing, data, images, television, home shopping, and services not yet conceived.

Consequently, the computer switches will need much more capacity, too. The same argument applies to local area networks and metropolitan area networks. One of the capabilities that will be incorporated on these switching chips will be the logic to interface to the high-performance network.

Most users will be hooked up to networks and use this interface logic. Most likely even if you are a hermit holed up in a cave with a personal computer, your computer will contain a chip with all these interface capabilities. It will be less expensive to include what most users want on a 50-million-transistor chip than to produce a separate line of chips for hermits.

Neural networks. This technology is just beginning to be implemented on chips. So far most neural-network applications have been simulated by programming on conventional digital computers. This approach enabled experimenters to

learn something about applications that neural networks could accomplish, but performance, of course, was very slow.

In the last few years several laboratories constructed neural networks in hardware. Carver Mead, Michael Emmerling, and Massimo Sivilotti at the California Institute of Technology fabricated 22 "neurons" on a chip. The neurons are actually a feedback network of operational amplifiers. At present, several companies market simple neural-network chips. For example, Intel's chip features 64 neurons connected to each other and to 64 inputs through 8,192 synapses.

One expects that the hardware embodiments of neural networks will increase in density during the 1990s, along with other types of processors implemented on silicon. Hardware versions should execute thousands of times faster than conventional simulations.

There are challenges ahead, however. One challenge is the steep learning curve demanded of development engineers who implement neural-network applications. In the early years of microprocessors, Robert N. Noyce, one of the founders of Intel, once remarked that the manuals sold better than the chips! Engineers had to learn how to use them. Neural networks will go through that stage, too, John J. Hopfield of Caltech told me in an interview.

At least four dark horses in the research barn are preparing to challenge silicon technology.

A second challenge is the number of connections between neuronlike elements. "In the typical VLSI [very large scale integration] circuit, the output current of a typical transistor feeds two or three other transistors and receives its input from a similar number," Hopfield noted.[8] "In neurobiology, that number is on the scale of 3,000, rather than 3, [which is] a qualitative difference."

Of course, the number of connections between neurons in current artificial neural networks is nowhere near the number of connections in biological neural networks. Even so, say researchers at Arizona State University, "the area required to route connections and to contain the average length of interconnections increases at unacceptable rates as more processing elements are added."[9]

A third challenge is to reproduce the learning capability of a biological network in an artificial neural network. Presently, artificial networks "learn" by setting the strengths of interconnections between the operational amplifiers. This learning occurs in a separate phase distinct from operations "by running through a massive database of information on the problem under consideration," Hopfield points out.[10] Biological systems learn on the fly while also continuing to perform. They also require much less information for learning than artificial networks.

Finally, current artificial neural networks are very simple compared with natural networks. The biological brain includes many capabilities that neurobiologists have not yet sorted out. Sorting them out is both difficult and time-consuming. The task won't be finished by 2000.

Dark horses

Meanwhile, at least four dark horses in the research barn—or just out of it—are preparing to challenge silicon technology. The farthest out in front is gallium arsenide (GaAs) since relatively small digital chips have been commercially available for five or six years. Computing with superconducting, optical, or molecular elements is still in the research stage.

GaAs. GaAs technology offers "clock rates between five and 10 times higher than CMOS [complementary metal-oxide semiconductor] capabilities," according to Ron Cates.[11] However, he said CMOS using less power "will remain the clear choice for most digital circuits." That still leaves the high-performance end of the marketplace for GaAs.

At the board level, GaAs devices work in conjunction with silicon chips to implement circuit paths in which propagation speed is critical. "In 1985, a commercially available GaAs circuit containing a few hundred transistors was rare," Cates said.[12] In 1990, "GaAs integrated circuits with over 100,000 transistors are mass produced and found in mainstream computer and telecommunications systems."

Continuing improvements in GaAs process techniques indicate a doubling of device density every nine to 10 months.—Cates

He expects microprocessors fabricated in GaAs to appear in the early 1990s. For some years to come, then, if the marketplace welcomes them, GaAs chips may follow a growth law comparable to the silicon pattern.

Superconducting. Two enticing lights appear at the end of this tunnel. First, superconducting Josephson junctions switch about 1,000 times faster than silicon transistors. Second, the heat dissipation of superconducting circuits is very low, which permits them to be packed tightly, reducing transmission time between circuits. Thus, "superconductors might be used to develop computers that would operate a million times faster than today's fastest computers," according to a 1990 report of the National Commission on Superconductivity, established by the US Congress in 1988.[13]

Of course, there are several serious scientific difficulties between here and there. That is why superconducting for computer applications remains in the research stage and promises to stay there during the decade. A handful of

continued on p. 68

Dedicated Tools for Microprocessor Education

Teaching often-abstract microprocessing concepts and applications in concrete terms is a constant challenge. To implement a practical curriculum, a team of educators developed several original tools for student use, including logidules, the CALM language, and the Dauphin and Smaky systems.

Jean-Daniel Nicoud

Swiss Federal Institute of Technology

Mastering such concrete objects as microcomputers takes practice. Instructors know the right amount of theory administered at an appropriate stage of expertise provides students with general knowledge and good working habits. Students also need access to plenty of computing equipment, which is often a problem for schools. These facts encouraged us to develop dedicated hardware and software tools for our minicomputer and microcomputer courses (see box).

Microprocessor courses

The minicomputer and microcomputer revolution convinced our school to create a computer science department. Students from various areas of a four-and-a-half-year curriculum can take a range of computer courses.

At this time, two courses on microprocessing are open to students: an introductory course on microprocessor software and interfaces for third-year students, and an advanced fourth-year course on microprocessors. Students carry out projects (10 hours per week) during the fourth year (seventh and eighth semesters). Finally two intensive months of diploma work conclude the curriculum. During this time students, under the supervision of a professor, develop a theoretical and/or practical engineering project, such as a board for a new microprocessor.

continued on p. 16

Development of student tools

In 1972 Laboratoire de Microinformatique (Lami) started with two Nova minicomputers—clearly not enough for the number of students using them. We asked Rene Sommer (a student who later developed the first mouse containing a microprocessor and designed all the versions of the Logitech mouse) to build a Nova front- and back-panel emulator[1] as a student project. Subsequently, the lab used 12 of these emulators until microcomputers became available.

As a result of this project, a Nova 1200 with only 4 Kbytes of program memory controlled a six-line local network of up to eight Novasims (see Figure A). Each line provided the feel of a real Nova. One could load a program on a

0272-1732/91/0200-0014$01.00 © 1991 IEEE

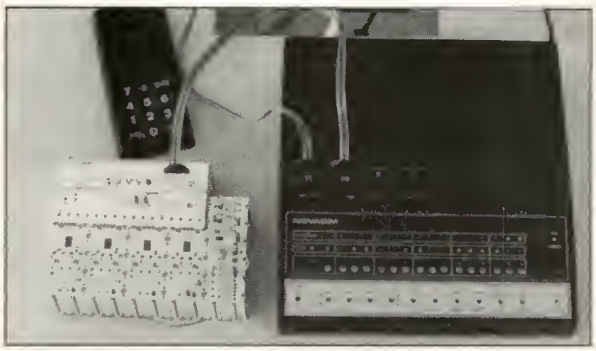


Figure A. A Nova minicomputer and a Novasim simulator.

front panel and execute step by step or at "full speed" (1,000 instructions per second). In addition, several I/O plugs equipped with a serial interface interconnected the lab peripherals uniformly.[2] Serial/parallel adapters provided the facility to build and control simple interfaces. We used "logidules" to construct these interfaces.

When microprocessors became available, we considered incorporating them inside the logidules[3] I had been developing (see Logidules box). In 1974, Dominique Dutoit built a microcomputer system[4] based on five "microdules" (see Figure B). Logidule boxes held automatic contacts for power supply and important signals. The bus did not contain enough automatic connections, so we used flat cables and 16-pin socket connectors. It looked beautiful, but it was not adequate for education since students were unable to work with the individual boxes. It also wasted time to reinsert connectors and try again if the system didn't work after installation.

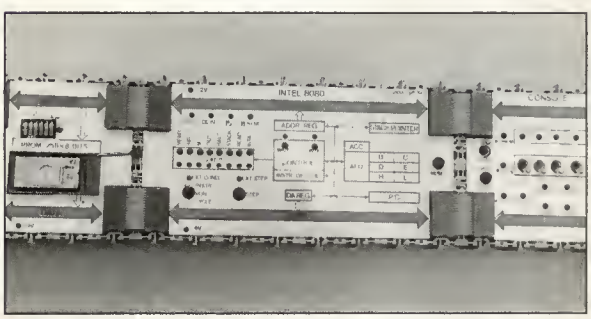


Figure B. Example of microdules.

In 1974 I used this same microdule bus to build the Portable Computer System (PCS) at Digital Equipment Corp. in Maynard, Mass. It featured an 8080 microprocessor, 4-Kbyte RAM, and a 16 × 64 alphanumeric screen. After signing a contract with DEC, we started to develop the line of Smakys (for Smart Keyboards).[5-6] We proposed specific interface solutions for personal computers.[7] Dante Del Corso at Politecnico di Torino also worked with us to define and promote the standard microcomputer bus, Mubus,[8] which we continue to use in its simplified 20-line fonn (Mubus 20).

Developing the first Smakys for our educational requirements was a long process. The objective was clear: the Smakys would include a local network made up of a small number of ICs and a Nova server with the huge 2 × 65-Mbytes disk we had on hand. Centralized printing and paper punching on the server was a priority. In the beginning, we centralized important programs like assembler, but we soon rewrote these programs for the Z80-based Smaky 6 stations.

While Sommer and others worked hard on this project, students learning Z80 assembler used the Novas to assemble programs. We loaded punching tapes onto a Dauphin system using hand-driven paper-tape readers we developed. (A description of the Dauphin system appears in the main article.) We used a large number of these low-cost Dauphin units because of their efficiency; they replaced the Novasim for teaching hardware interfaces (see Figure C).

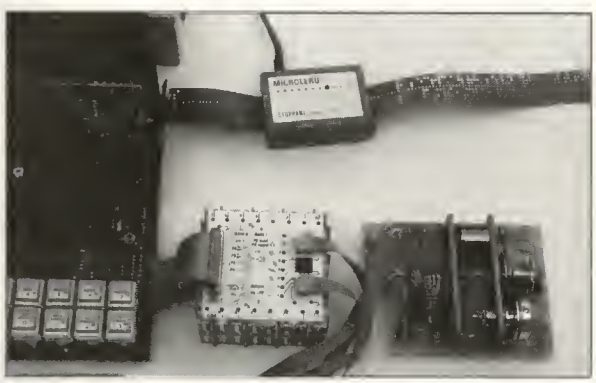


Figure C. Dauphin system with miniature paper-tape reader and logidules.

The first step[9] towards developing our Common Assembly Language for Microprocessor (CALM) also occurred at DEC. Rick Merrill created a cross-assembler on a PDP 11 to make Intel's notation of the 8080 assembly language more coherent. Still in use, adequate tools now support it.[10]

continued from p. 14

Presently, 100 students receive exposure to microprocessors. However, only an average of 10 students who complete *all* the courses, projects, and diploma work understand enough to be efficient in working on microprocessor-hardware-related industrial projects.

Figure 1 shows the organization of the curriculum. A textbook[11] provides the foundation for introductory course. Instruction for the first term (fifth semester) begins with hexadecimals, two's-complement, and binary operations. An emphasis on mnemonics and operands familiarizes students with assembly language notation before discussing von Neumann architecture. Students spend a few hours using the Dauphin system (detailed later) to practice selected hardware and software principles. Additional instruction focuses on addressing modes and assembler syntax.

After a few exercises[12] students select an individual program to write; they use a complete library of routines to write rather spectacular programs.

Course frequency reduces during the second half of the term. Easy access to the lab encourages students to spend extra time in practical work. The network of Smakys we use has been successively based on Z80, 68000, and 68030 microprocessors (see box).

We devote the sixth semester to hardware instruction. We require computer science students to learn about building controllers using standard ICs or gate arrays. They take regular lab courses[13] and pick up as much as they can from the course, which will soon be available as a book published both in French and in English.[14-15]

The fourth-year Microprocessor course is an elective for electrical engineering, computer science, and microtechnic students. (Microtechnic students study mechanical engineering with an emphasis in electronics and fine mechanics, such as robotics and micropackaging.) Strongly hardware-oriented, the first term (seventh semester) covers basic microprocessor features. Course content includes bus interfacing, interrupt handling, programmable interface understanding, and microcontroller knowledge. After studying basic concepts theoretically, students practice applying these concepts inside several processors.

This course often proves difficult for the students, who interact for the first time with the many technological and timing constraints related to the correct operation of a microprocessor surrounded with memory and I/O devices.[16] Although the material is available[17], we only briefly survey buses[18] because of lack of time.

The final term (eighth semester) covers modern microprocessors like M68030, 486s, reduced instruction-set computers (RISCs), transputers[19] and digital signal processors. It also presents video RAMs[20] and sophisticated I/O controllers for color displays, Ethernet, and Small Computer Systems Interfaces (SCSIs). The content varies from one year to another.

With each year, we continue to learn how to streamline the course. Now we can discuss a 16-bit processor in the seventh term and the 32-bit processor, caches, memory

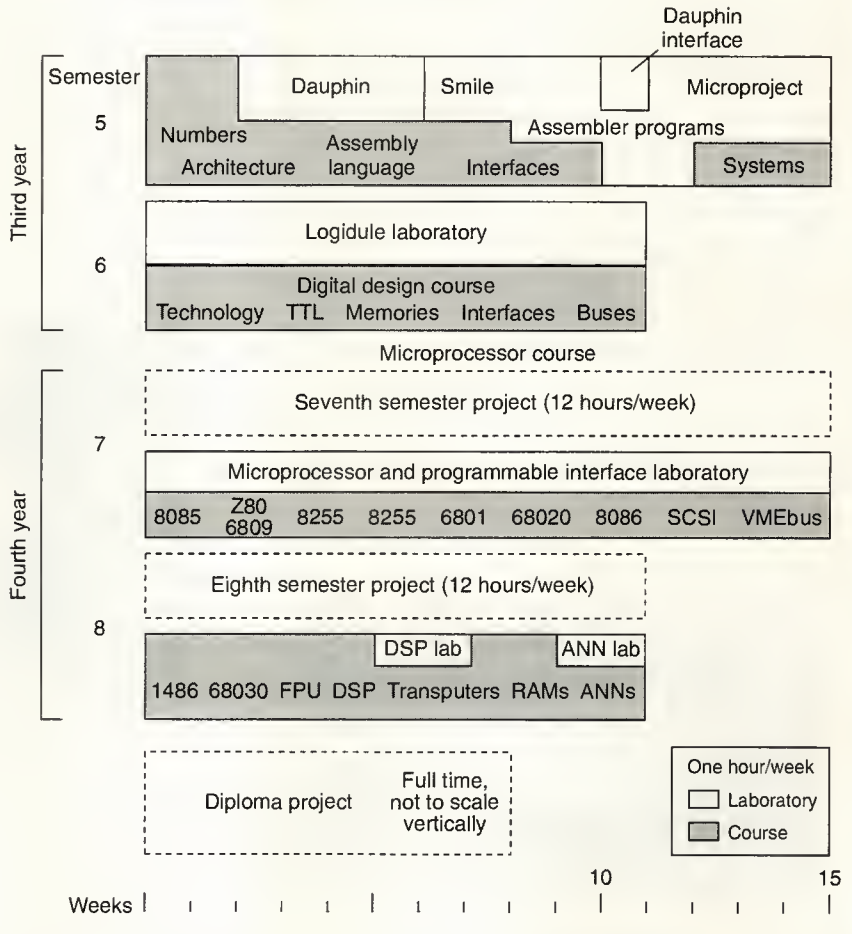


Figure 1. A proposed microprocessor course curriculum.

Why more powerful PCs for education?

The design of the Smaky 6 occurred at the same time as that of the Pet, Apple II, and Tandy computers. The local network connecting the Smaky 6 only required 10 ICs; it was probably the first low-cost local network for microcomputers.

In 1984, the Smaky 100 replaced Smaky 6, with its Z80 microprocessor, to make the most of the 68000 architecture. Since the new lab needed more machines in 1989, we decided to develop a new 68030-based machine, the Smaky 196, instead of increasing the number of Smaky 100s. The decision to change the machine was not really prompted by the assembly language course; a 68000 was powerful enough. The Smaky 196 68030 enabled us to add a MMU and a FPU lab for the Microprocessor course. In both cases, the machines have no disks, which simplifies maintenance considerably as well as the follow-up of students' activities. A comparison of the Smaky 6 and Smaky 196 systems appears in Table A.

Note the difference in student productivity over the last nine years. Many benefits resulted from the improved system performance, enhanced understanding of the microcomputer field, and early exposure to computing through home computers and discussions with friends. Students accomplish more complex work: they handle larger programs, use a greater variety of system calls including character generators and sprites, and produce better documentation.

The main consequence of this greater productivity is increased disk usage. The solution is relatively simple since a classroom of 60 students "only" needs 15 Mbytes.

Table A. Comparison of Smaky system features.

Feature	Smaky 6	Smaky 196
Start of development/ operational	1978-80	1989-90
Number in classroom	20	40
Alphanumeric screen	16 × 64 characters	According to character generator
Graphic screen	256 × 128	640 × 400
Mouse	Option	Yes
Processor	Z80, 4 MHz	68030, 20 MHz
ROM/RAM	4 Kbytes/32-64 Kbytes	256 Kbytes/1-4 Mbytes
FPU	None	68882
Extension bus	Mubus 74	68030 lines
I/O bus	Mubus 20	Mubus 20
Parallel/serial port	16 bits/2 UART	None/SCC
Local network	120 Kbits/s	500 Kbits/s
Disk server/size	DG Eclipse/130 Mbytes	Smaky 196, 140 Mbytes
Printer server/printer	Same Eclipse/Versatec	Smaky 324/Laser
Operating system	Self, monotask	Self, multitask
System calls	50	40 (+ 300)
Editor/assembler	Smile (CALM 2)	Smilec (CALM)
Other programs	Basic, UCSD	Basic, Pascal, Modula, C
Utilities	Editor	Macintosh-like environment
Average student disk usage (bytes) sources/images/ character generation	4 Kbytes/1Kbyte	12Kbytes (maximum of (55 Kbytes)/50 Kbytes/ 20 Kbytes

management units (MMUs), and RISCs in the eighth semester. We introduce artificial neural networks (ANNs) at this time, and students can work with a powerful set of simulators.

During the two semester projects and the diploma work, students may elect to conduct a hardware project. They receive a processor, microcontrollers, or a programmable interface, from which they can design a system or interface, or solve a particular application.

CALM development

The objective of a microprocessor course is to understand the possibilities of both simple and sophisticated processors. Preparing the students adequately requires writing several programs for different processors and system routines. Clear and consistent notation simplifies and accelerates this understanding. Manufacturers tend to use implicit operands and short mnemonics. This tendency saves a little time for experienced

continued on p. 62

Fun and Games and Microcomputer Interfacing

It is one thing to teach computer interfacing within a formal course structure. Making it interesting and educationally sound for the computer science/engineering student is another matter. Both aims can be achieved and mutually support one another. In the microcomputer laboratory described here, students learn computing concepts as they compose music and control slot-car and model railway sets.

John A. Fulcher

University of Wollongong

A sound theoretical introduction to fundamental principles is the foundation of most computer science/engineering courses on computer interfacing.[1] However, such fundamental concepts are a waste unless laboratory exercises reinforce them.

In these exercises, students can experience in an immediate and tangible manner the fruits (and sometimes follies) of their programming efforts, by seeing lights flash, motors turn, and solenoids operate.[2] Moreover, such hands-on laboratory work can hold its own intrinsic interest, stimulating the students' imagination. Such stimulation ensures that the educational principles underlying the various laboratory exercises hit home more effectively.

Laboratory setup

At the University of Wollongong, we have developed a microcomputer laboratory to support the third-year Microcomputer Interfacing and Real-Time Computing course for computer science majors.[3-4] The exercises students perform in this laboratory include:

- a transparent serial communications link (different from the on-board link on the 68KECB single-board computer);
- generation of computer music;
- decoding patterns from a bar wand;
- low-resolution, bit-mapped graphics;
- control of a turtle robot;
- low-level control of a floppy-disk drive (and the subsequent writing of disk-handler software);
- movement of an *x-y* drill positioner;
- data logging (including statistical analysis of the readings taken);
- control of a slot-car set; and
- model railway scheduling.

Figure 1 shows the Microcomputer Laboratory setup: to the left of the terminal keyboard is the RS-232C D-type connector from the host Unix computer. A Motorola 68KECB single-board computer[5-6] is housed inside the cabinet on the right side, together with its power supply and the locally developed front panel. This front panel provides additional, simple I/O devices to assist in testing and debugging control software.[3]

A plug-in experiment pod connects to the 68KECB via a 50-pin socket on the front panel. This pod comprises a printed circuit board that replaces the lid of a black plastic box. A range of such plug-in experiment pods is available. Each experiment has its own dedicated plug-in jiffy box. Thus, laboratory setup for different experiments simply involves unplugging the existing jiffy box and plugging in another.

We constructed all the necessary interfacing hardware for the students and housed it within these jiffy boxes. We built the Microcomputer

0272-1732/91/0200-0018$01.00 © 1991 IEEE

Figure 1. Computer equipment in the Microcomputer Laboratory.

Laboratory to support computer science students, rather than computer engineering students. Therefore, they must have a reasonable background in high-level language programming from earlier courses, but they can have little or no hardware background.

The 68KECB firmware contains a transparent link communications program that allows users to develop MC68000 assembly language (or C) programs on the Unix host. Users can compile them using the Unix cross-assembler (C compiler), then download (in Motorola S-record format) the resulting executable file to the target 68KECB via the RS-232C link.

We expect students to program the various peripherals via the plug-in interface pods supplied to them. Since a memory-mapped system is involved, students write data bytes to a control register (one memory location), read bytes from a status register (another location), and read/write data to or from a data register (a third memory address). However, we believe an understanding of how the peripheral actually works is also desirable, and this is presented to the students during the formal lecture portion of the Microcomputer Interfacing and Real-Time Computing course.[1]

During the first few weeks of the laboratory segment of this course, students familiarize themselves with the MC68000 instruction set; addressing modes; Unix cross-assembler and 68000 C compiler; and simple, memory-mapped I/O devices on the front panel. After this familiarization period, they "dive in at the deep end" and begin writing control software for the real-world peripheral devices (see the next section). Thus, students are given the choice of developing their software in either 68000 assembly code or C.

Laboratory exercises

All students conduct the same experiments for the first half of the laboratory portion of the course. For the remainder of the course they choose between a range of special interest laboratory exercises.

Some of these exercises are deceptively simple at first sight, yet in reality they mask several sophisticated concepts. For example, timing periods are critical in the bar code experiment, and the turtle robot requires real-time control (with feedback provided via the robot's bumper sensors). In addition, students can incorporate learning algorithms into the slot-cars' controller, and high-level scheduling algorithms in the model railway. McKerrow[7] previously demonstrated the educational value of these latter "games." Students have fun and at the same time learn important real-time programming principles!

Computer music. Figure 2 shows how all three timer/counters within an MC6840 Programmable Time Module (PTM) generate computer music. For better coloration in moving up or down the scale, we use a fixed mark/variable space technique rather than simple square-wave music. With this technique, the same score played in different octaves sounds as if it is being generated by different instruments. Square-wave music, by contrast, produces electronic organ-like sounds across all octaves.[1]

Counter 1 produces a fixed-width, one-shot pulse at regular intervals (or pitch) as determined by counter 2. Counter 3 determines the duration of a sustained note. We encode the musical score in the form of a lookup table of pitch/duration pairs, with a second lookup table that converts pitch data into timing values for loading into counter 2 (see Tables 1 and 2 on the next page).

Students play either their favorite tune or song that best summarizes their experience in the third-year course. The resulting efforts range from simple nursery rhymes to two-part Bach concertos (two MC6840s are on the music experiment pod to serve those students who feel so inspired). Obviously, timing is critical in this exercise, and students become quite proficient in programming the MC68000 for interrupts as a result.

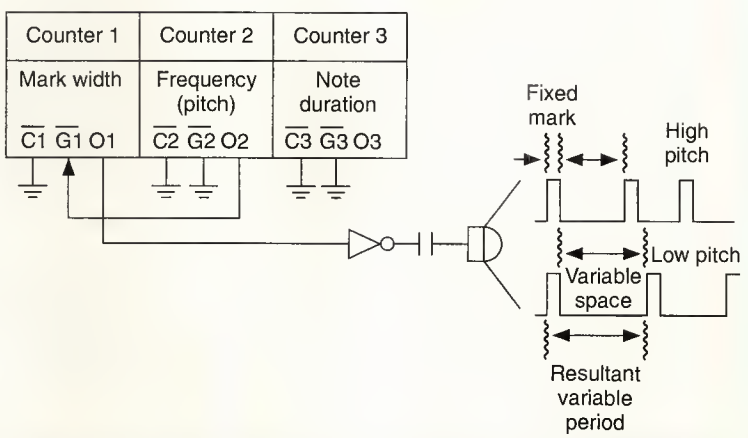


Figure 2. Diagram of music generated by computer.

Table 1. Lookup table: pitch/duration.

Note	Duration
A	0.5
Bb	1.0
⋮	⋮

Table 2. Lookup table: pitch data.

Tone	Frequency	No.of program loops
A	440Hz	31
⋮	⋮	⋮

Figure 3 contains a complete music program listing. The main program simply loops indefinitely until MC6840 PTMs interrupt it. The 6840 generates accurate timing intervals, which interrupt the CPU at the end of each interval. Students also gain an appreciation of the waveshaping capability of the PTM support chip. Multitasking is required when using both 6840 chips on board for the plug-in experiment (to play two-part musical pieces).

Analog-to-digital conversion. Figure 4 shows a diagram of the analog-to-digital experiment pod. The pod holds an A/D converter and D/A converter, both of which allow the switching of I/O to two different sources, or destinations. Software control accomplishes this switching via the MC6821 Parallel Interface Adapter (PIA) by using analog metal-oxide semiconductor (MOS) switches.

We can connect the high-precision, 10-turn potentiometer directly to the meter, or alternatively to the input of the A/D converter. Signal conditioning can be applied to the raw readings prior to subsequent display on the meter. Then we can switch the D/A converter output to the meter or input it to the A/D converter.

In this experiment, we alert students to the need to compensate for offsets and nonlinearities in the potentiometer, as

```
;*******************************************************************************
INIT:       move.b      #$80,$30403     ;select Control Register#3
            move.b      #$43,$30401     ;enable ints (counter#1)
            move.b      #$81,$30403     ;select Control Register#1
            move.b      #$A3,$30401     ;set counter#1 as oneshot
            clr.b       $30405          ;load preset value = $0050
            move.b      #50,$30407      ;-> counter#1 (fixed mark)
            lea         PITCH,A0        ;pointer to pitch table
            lea         SCORE,A1        ;pointer to score table
            move.w      #$2300,SR       ;allow interrupts > level-3
;*******************************************************************************
MAIN:       move.b      (A1)+,D1        ;get first note to be played
            jsr         PLAY            ;and play it
            move.b      #$A2,$30401     ;start all 3 counters
WAIT:       bra         WAIT            ;loop until interrupted
;*******************************************************************************
INTSRV:     move.b      (A1)+,D1        ;get next note to be played
            cmp         #$FF,D1         ;test for score termination
            beq         EXIT            ;if so, stop playing music
            jsr         PLAY            ;otherwise play it
            rte
;*******************************************************************************
EXIT:       clr.b       $30401          ;disable further interrupts
            rte
;*******************************************************************************
PLAY:       mulu        #2,D1           ;index into pitch table
                                        ;(of words, not bytes)
            move.b      0(A0,D1),$30409 ;MSB of pitch ->counter#2
            move.b      1(A0,D1),$3040B ;LSB of pitch -> counter#2
            move.b      (A1)+,D1        ;second byte = duration
            mulu        #5000,D1        ;absolute tempo (# beats
                                        min.)
                                        ;(could make user-variable)
```

Figure 3. A complete music program listing (continued on next page).

well as to calibrate the meter initially for full-scale deflection. We use a front panel to initiate successive readings of the potentiometer, and another switch selects direct or complementary (10 volts-V_t) readings. Students can store sets of up to say 10 readings for later display on a dot matrix printer connected through the Centronics port of the 68KECB. They can also calculate minimum, maximum, and average readings using their own rudimentary statistical analysis routines.

Once again, interrupts play an important role in this experiment, but more importantly students confront real-world "nasties" such as transducer nonlinearities

continued on p. 75

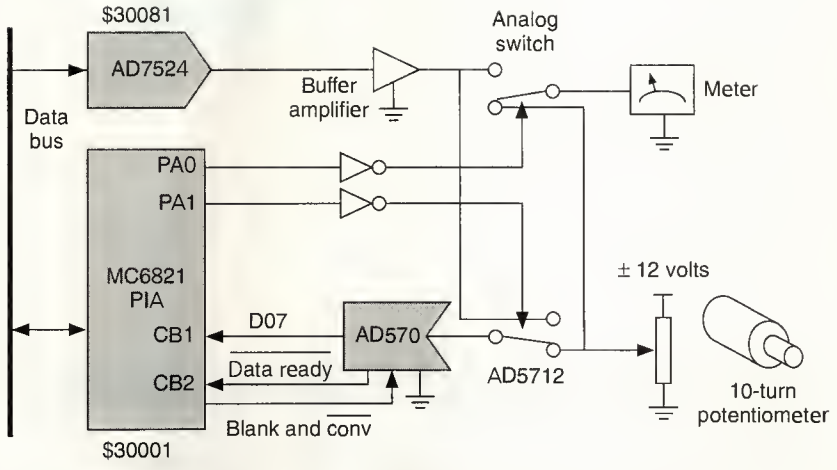


Figure 4. Analog-to-digital experiment pod and the conversion process.

```
                    move.b          D1,D2           ;need to separate bytes to
                                                    ;send to 16-bit peripheral
                    asr             #8,D2           ;over 8-bit bus
                    move.b          D4,$3040D       ;MSB duration -> counter#3
                    move.b          D5,$3040F       ;LSB duration -> counter#3
                    move.b          #$A2,$30401     ;start counting down
                    rts
;*******************************************************************************
    PITCH:          DC.w            0,1926,1818,1716,1620 ;rest, Ab, A, Bb, B (0-4)
                    DC.w            1529,1443,1362,1286   ;C, C#, D, Eb (offset = 5-8)
                    DC.w            1213,1145,1081,1020   ;E, F, F#, G (offset = 9-C)
                    DC.w            963, 909, 858, 810    ;Ab, A, Bb, B(D-10)
                    DC.w            764, 722, 681, 643    ;middle-C, C#, D, Eb
                    DC.w            607, 573, 541, 510    ;(261.6Hz) E, F, F#, G
                    DC.w            482, 455, 429, 405    ;Ab, A, Bb, B (19-1C)
                    DC.w            382, 361, 341, 321    ;C, C#, D, Eb (1D-20)
                    DC.w            303, 286, 270, 255    ;E, F, F#, G (21-24)
                    DC.w            241, 227, 215, 202    ;Ab, A, Bb, B (25-28)
                    DC.w            191, 180, 170, 161    ;C, C#, D, Eb (29-2C)
                    DC.w            152, 143, 135, 128    ;E, F, F#, G (2D-30)
                    DC.w            120, 114, 107, 101    ;Ab, A, Bb, B (31-34)
                                    :
                                    :
;*******************************************************************************
    SCORE:          DC.w            $1504, $1504, $1508, $1504, $1504, $1508,
                    ;jin_gle bells, jin_gle bells, (E E E, E E E)
                    DC.w            $1504, $1804, $1104, $1304, $1516, ...
                    ;jin_gle all the way... (E G C D E...)
                    :
                    :
        ;(DC.w $FFxx to terminate)
        ;4 = semiquaver, 8 = quaver, 16 = crotchet, 32=minim ...
```

Figure 3. (continued.)

An Advanced Educational Microprocessor System

The host-independent, 68030/68882 AEMS board is useful as a classroom demonstration tool as well as a laboratory instrument. In a hands-on approach students learn about microprocessors and various topics that use many of the features of the system, including digital signal processing, digital control, and image processing.

L. Howard Pollard

Ramiro Jordan

University of New Mexico

One of the great challenges facing instructors today is to effectively teach about and with microprocessor systems. The University of New Mexico developed a board that facilitates this process. This tool helps us explain microprocessors and computer-related concepts and systems to students. We also use it for other portions of the curriculum to demonstrate other concepts as well. The board's state-of-the-art components and facilities make available to students all of the appropriate signals. In addition, the board uses features that let students examine concepts related to computer architecture, data networks, operating systems, digital signal processing and control, and a number of other areas.

The Advanced Educational Microprocessor System acts as a demonstration vehicle as well as a laboratory instrument. To facilitate its use in both of these areas, we incorporated features in the system that are not available in many other educational processor implementations. To demonstrate the interaction of all of the portions of the AEMS, we made the buses and control signals readily available, inviting the easy connection of test equipment, such as logic analyzers and oscilloscopes, to the system. This facile accessibility to signals and buses allows students to observe the interaction of the different elements that make up the system and encourages a hands-on approach to education.

In addition to the easily accessed signals, other AEMS features provide additional educational opportunities: counting events, monitoring activities, and observing system activities. These features also allow students to configure many of the characteristics that impact system performance. In addition, students can easily include devices of their own to tailor experiments to an application or implementation technique.

0272-1732/91/0200-0022$01.00 © 1991 IEEE

Concurrent to the implementation of the board, we are developing course materials to facilitate the use of the system in a variety of courses. Of particular interest for us are courses in computer engineering, such as programming techniques, processor architecture and design, operating systems, and data networks. Within the electrical engineering curriculum, we are developing course materials that use the board in digital signal processing, digital control, robotics, and other related areas.

The AEMS board compares favorably with the ECB68000, the Educational Computer Board available from Motorola, Inc. Based around the 68000 processor,[1-4] the ECB68000 contains two RS-232 ports and a parallel port. Its system memory is limited to 32 Kbytes of dynamic RAM and the Tutor firmware,[1,4,5] which is contained in 16 Kbytes of ROM. The Tutor firmware lets students perform basic input/output operations as well as examine and modify memory locations. With these operations, students can test code, assemble and disassemble simple code, and perform other limited operations. However, the ECB68000 board does not contain an Ethernet capability, event counters, bus interfaces, floating-point operations, or cache memory.

The AEMS

A block diagram of the Advanced Educational Microprocessor System appears in Figure 1. At first glance, this system does not seem specifically targeted to education nor flexible in its use. We will take each of the features indicated in the figure and describe it and its educational impact.

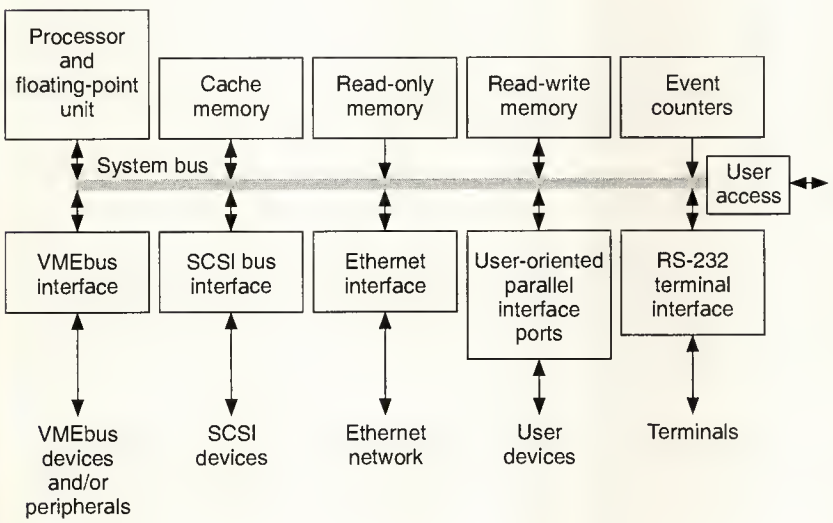


Figure 1. Block diagram of the Advanced Educational Microprocessor System.

Processor and floating-point units. In the current version of the AEMS a Motorola 68030 processor[6,7] and 68882 floating-point unit[6,8] provide a highly effective computational system that is capable of both integer and floating-point activities. Important in applications involving digital signal processing and digital control, these capabilities let students studying concepts such as stability, steady-state errors, and noise reduction easily investigate the effects of finite-precision arithmetic. We've found that students learn the theoretical design concepts best when exposed to real-life problems such as truncation and round-off errors.

Under planning now are two versions that will incorporate the 56000 DSP and 88000 RISC processors. The capabilities of these computing engines sufficiently support almost all needs in educational applications, both in course work and research.

Memory system. Cache memory has become a feature included in almost all processing systems, regardless of size. The cache included in the AEMS includes 256 Kbytes of accessible memory. Students can configure the size of this system (from 32 Kbytes to 256 Kbytes) so that studies and demonstrations may be readily carried out. In addition, students can easily count events associated with the cache (read, write, hit, miss), and thus, can carry out a performance analysis on the system. Currently, we use a write-through cache system. However, one of the experiments with the board is to implement a write-back protocol for the system.

The read-only memory includes those portions of the system that are required in permanent storage. In particular, this portion of the memory contains code for downloading programs from a remote host, as well as a version of the Tutor monitor. The download capability uses the TFTP (Trivial File Transfer Protocol) method of accessing programs that are available to the board over the Ethernet system. In this fashion, the board remains independent of its host as long as a TFTP server is available via the Ethernet connection. The Tutor monitor allows students to control activities within the system on a real-time basis and to examine registers and memory, assemble and disassemble code, and make changes as needed.

The read-write memory in AEMS consists of eight packages, each containing a megabyte of storage, plus the logic required to control the memory. Since this is dynamic RAM, the control consists of a refresh timer plus the logic required to appropriately feed row and column addresses to the DRAM chips

themselves. The cache memory and DRAM must cooperate in a very tightly controlled fashion, so the interaction of addresses and data must be performed in a rigorously controlled manner. The 8 Mbytes of storage suffice for most applications; however, if more storage is required, students may use additional DRAM chips.

Event counters. One of the very useful facilities of the AEMS for education and research is a set of event counters. This set of four counters lets students trace any event of interest over time. These counters can be cleared and read, letting students monitor occurrences of events over a period of time. Students can (independently) derive the counter clocks from a constant frequency (for a timer application), from cache events, from events generated by the processor itself, or from student-supplied events. Note that these counters are part of the hardware of the system and consume no system resources (time) to monitor any events. Hence, a student can set up an experiment, including establishing hardware conditions and resource allocation as appropriate, and after clearing the counters, initiate the action to be monitored. The event counters will continue to monitor the activities identified by the student until the experiment is complete. The only system resource needed is the time to read the counters when the experiment is over. This feature of the system is important for carrying out performance analysis on a variety of systems.

I/O interfaces. Several mechanisms in the AEMS connect to the outside world. In a stand-alone mode, two RS-232 terminal ports allow access to the Tutor monitor and through it access to the facilities of the system. When the proper programs are running, two users can access the system facilities.

Another interface included with the AEMS is the Ethernet network connection, which allows communication with local area networks in which a variety of work is accomplished. The network connection also allows access to any peripheral (a disk or printer) that is available on the network. Code included in the resident ROM will load into the AEMS memory over the network connection any program that is available from a TFTP server. Hence, students can develop a program on another system, if desired, and load the binary version into the AEMS for execution.

If students desire to use the general-purpose peripheral devices of the educational system, they can easily do so in two ways. One method involves using the SCSI (Small Computer Systems Interface) connection. For example, a SCSI disk system could be connected, providing mass storage for operating system interaction or data storage requirements.

Another mechanism is a VME interface. Although the board is longer than "normal" VME interface boards, its design lets it be connected into a VME system. Hence, the AEMS can access any of the peripherals or other facilities available over the VMEbus.

Another capability of this computer board is its parallel port interface, which allows students to connect devices needed for experiments and demonstrations directly to the system. Students can also incorporate analog-to-digital and digital-to-analog converters into the system for digital signal processing and digital control applications. We encourage students to create a printer port, or to capture data from a real-time source, as part of interfacing exercises. Only the imagination of the student and the availability of appropriate parts limit the use of the parallel port.

User access. We labeled one box in Figure 1 simply "User access." This box indicates that all signals associated with the bus, as well as most of the other devices of the system, are readily available to the student. Students can access signals on pins at the edge of the board and directly connect them to logic analyzers or oscilloscopes. Students can then use these signals to provide visual and printed information so they assimilate the details that accompany the concepts presented in course material. In addition, a student can interface to the system in a very direct way. The important aspect of this feature is that the signals are neither buried nor hidden from the student. Rather, we encourage students to access the signals and study the characteristics of the system to achieve a hands-on approach to education and training.

Not shown in Figure 1 are a number of relatively minor features that we have incorporated into the system, features that allow the system even more versatility. For example, the construction of the board's interrupt system lets students easily change the priority of the interrupts associated with devices in the system. This feature gives insight into the issues involved with certain I/O operations. Also, the extensive programmable logic used in creating the required "glue logic" lets students who want to change some of the characteristics of the system easily substitute a different set of PLAs (programmable logic arrays). In particular, the DRAM controller is implemented with a programmable state machine, and, hence, the control of the DRAM and its interaction with the cache can be modified without making extensive changes.

The result of the features just enumerated is a board that can be used in a variety of applications in education. We now look at some of those applications.

Incorporating the AEMS into the curricula

Because of the flexibility of the new educational computer board, teachers can easily incorporate it as a tool in the educational and research environments. We have used the board in several of the courses we currently teach and in research environments.

Computer engineering. In the computer engineering curricula, we plan to use the new tool in two courses, the microprocessor course and the microprocessor design lab. Figure 2 identifies the manner in which we connect the facilities to the AEMS for these courses. The user terminal provides direct access to AEMS facilities. The logic analyzer and oscilloscope

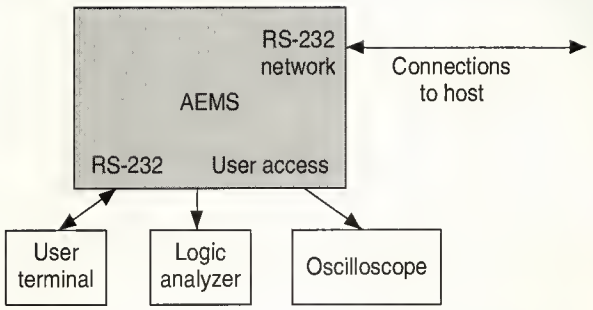


Figure 2. The AEMS as used in microprocessor courses.

provide windows into the system for observation and analysis. In addition to these aspects, we can easily interface simple experiments to the board.

The introductory course in microprocessors and microprocessor-based digital systems[9,10] exposes students to architecture and assembly-language programming, testing, and debugging. We cover in depth concepts such as data types, stacks, queues, fixed-point arithmetic, floating-point arithmetic, subroutines, and interrupts. We also introduce the hardware concepts of asynchronous and synchronous bus interfacing, addressing techniques, memory-mapped I/O, and serial and parallel interfacing issues. We emphasize the process of interfacing with external devices to acquire data, to process it, and to display the final results.

In the microprocessor design lab, students participate in a modular approach to system design. A group of students must design, test, and evaluate a microprocessor-based digital system as one course requirement. For the microprocessor design course, the AEMS can be an extremely useful tool for learning and comparison, because of its easy access to internal signals. Examination of the signals and their relationships helps students understand timing, circuitry for decoding, buffering, interrupt priority, and other issues. We require that students demonstrate a hardware/software project at the end of the semester; our board provides an excellent platform in doing so due to its inherent interfacing capabilities.

The AEMS board will be a very useful teaching aid in an operating systems course and a networking course. Figure 3 demonstrates one mechanism for connecting the AEMS for these courses. In addition to the user terminal, a disk connects directly to the AEMS. Students can use the logic analyzer and oscilloscope to monitor any appropriate signals. They can make connections to other systems via the Ethernet port ("network" in the figure). Since the AEMS is independent of its host, students can easily download routines developed in different machines for testing and evaluation. Students have the flexibility to assemble and disassemble routines through the use of the built-in monitor program (Tutor), as well as access resources through the Ethernet network and SCSI interface. Students can investigate resource management, real-time programming, distributed computing, concurrent processing, security issues, multitasking, and multiprogramming concepts. For instance, through the SCSI connection students can connect to a disk and perform an analysis on scheduling policies for disk interaction or page replacement strategies.

The Ethernet interface has become a standard feature in available state-of-the-art workstations. One of the reasons we included the Ethernet interface in the AEMS board was to give students an appropriate tool to monitor the network. Thus, its use in the data networks course is appropriate. With its easy access to internal signals and the flexibility of the internal event counters, the AEMS board forms an excellent tool for analyzing packets, investigating routing and flow-control algorithms, and providing insight into the use of local area networks. It also provides a convenient mechanism for visualizing the different layers involved in computer communication protocols and a vehicle for investigating data encoding/decoding techniques and the design of measurement and evaluation procedures. Because of the power and flexibility of the AEMS board, teachers can also use it to teach and perform research in the fairly new data transmission area of ISDN networks, in which speech and data are discretized and merged.

Electrical engineering. Because of its computing engine, the AEMS is a versatile tool for electrical engineering courses to teach digital signal processing, digital control, and

continued on p. 78

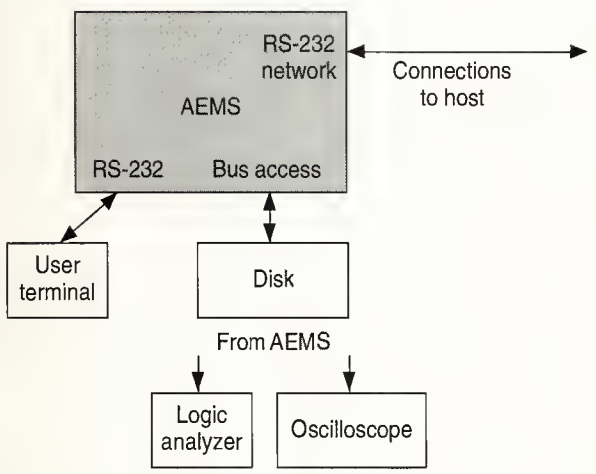


Figure 3. The AEMS in a typical operating system and network setup.

A Configurable, Virtual Microprocessor System

for Instructional Use in Real-Time, Real-World Studies

This system allows students to simulate and validate a process plant of their own design, associate I/O channels to the individual components of it, and write a control or sequencer program to control the plant operation using any given assembler. A cartoon display option shows the plant and a trace of the program in separate windows.

David W. Russell

Penn State Great Valley

Kirtley B. Haden

Leeds & Northrup

Instructors find real-time, real-world computer applications exceedingly difficult to teach in the classroom setting. The lack of appropriate industrial instructional laboratories or the limitations imposed by oversubscription to them leaves instructors in a quandary. To attempt to teach the subject on paper alone is as unsatisfying as a half-hearted, poorly equipped laboratory session. Laboratory rigs are inflexible, unreliable, and present interfacing difficulties. The student is left to ponder whether the experiment was really a valid experience that will actually relate to what will be found in the outside world. Graduate-level courses intensify the dilemma because the student may already be working with that reality on a day-to-day basis.

Courses in direct digital control can often be subdivided into three parts: system design and instrumentation, control, and actual operations. The instructional system we describe attempts to provide a valid learning environment using a personal computer, without the necessity for an automation laboratory.

Instructors can add a control section to this system that will produce random faults, power failures, and input/output errors in a simulation. With this capability, instructors can test a student's design for completeness, error handling, and fail-safe operation. Some self-correcting faults simulate a repair; the control program restarts as and when appropriate, while other conditions must close the plant down.

In the graphics mode of operation the plant design appears as a cartoon with fluid levels altering, pumps switching on and off, and so on, according to the control effected by the student program. The system also displays a trace of the execution and I/O status of the program under execution. If no process design is included, the system can be used to run simple assembler programs on a stand-alone basis. The system,[1] was written in response to a need here and at off-campus instructional facilities for a teaching tool usable in graduate-level real-time, real-world computing courses.

Figure 1 shows an overview of the system design.

0272-1732/91/0200-0026$01.00 © 1991 IEEE

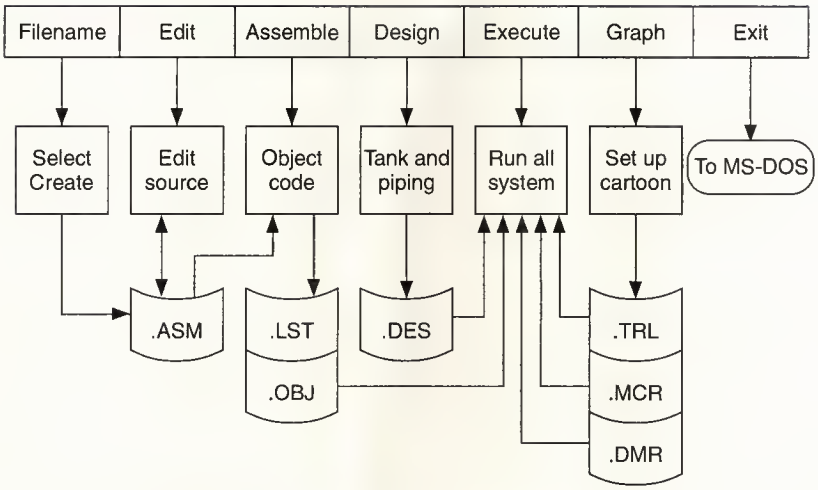


Figure 1. System overview.

Students may also configure the instruction set that the assembler uses, allowing the illustration of the varying richness of commercially available microprocessors. Since access to real I/O is often not standard, the system considers macro definitions such as ADC and DAC to be available via a common library. In a real situation, a user may be presented with a different mnemonic or forced to construct a macro at the operating-system level.

The system can run in one shot or single-step mode so that students can see the impact of every instruction one at a time—an established method of debugging such programs. The system also runs in slow motion, so the overall flow control of the simulated plant can be associated with the computer instructions that in reality execute too fast to perceive. We designed the menu-driven system to require minimum keystrokes to change modes and displays.

Process design and I/O configuration

When the simulator is to be used in control mode, the student selects the Design option on the master menu. This step allows the student to configure an arrangement of tanks, pumps, mixers, and heaters into a simulated plant. The Design option keys from the tank number. Upon selecting a tank, the student can enter or modify text that will be displayed in graphics execution mode. The student can assign initial minimum and maximum volumes to fluids within the tanks, pump rates and I/O channels, heater and mixer controls, and pipe routing. The I/O designations are referenced by the control program ADC/DAC/DIN and DOUT directives.

When the simulation is active, the volumes in the tanks reflect the pump flows integrated over the time window. This feature serves to emphasize industrial aspects, limitations, and the need for accurate instrumentation.

The student can also select a Graph option and design the plant for dynamic graphic display during execution. The selection produces a cartoon of the plant operation.

Assembler language system

The student must select a filename (which is given the .ASM extension) when entering a control program. This file constitutes the source code for the particular application. The student enters this program as follows:

```
(Label) op-code (OPERAND 1) (OPERAND 2)
    (* Comment)
```

where ( ) means the field is optional or dependent on syntax.

After entering the program source, the student exits from this Edit option by using the F10 function key. The source is simply a text file that must be linked to the simulation of the computer architecture and to the process design interface. The Assemble command converts the source code into an object file. Students execute this file by selecting the Execute option from the menu. The system can be run in single shot, slow motion, or full speed by invoking the Mode option. In execution the program trace appears on the screen so that the student can view the effects of each instruction on the process plant and observe the program flow control within the virtual microprocessor.

The Assemble command produces an object (.OBJ) file and a list (.LST) file for debugging purposes. Use of an I/O page brings the interface into the process simulation.

A simple example of system use

The following presents a simple example of our system. The object is to empty a tank of fluid (Mix) to some minimum level (M4) using the pump (P4). See Figure A.

The configuration appears in Table A.

The control program segment would be:

```
        LDI     M4 , r2  ; Set minimum level for tank
        LDI     1  , r3  ; Put 1 into register 3
        DOUT    #4 , r3  ; Use it to turn P4 ON
DRAIN,  PAUSE   10       ; Wait 10 time units (eg sec.)
        ADC     #4 , r1  ; Read Tank #4 level into R1
        SUB     r2 , r1  ; Compare with MIN level M4
        BPL     DRAIN    ; Repeat if level above M4
        CLR     r3       ; Clear register 3
        DOUT    #4 , r3  ; Use it to turn P4 OFF
```

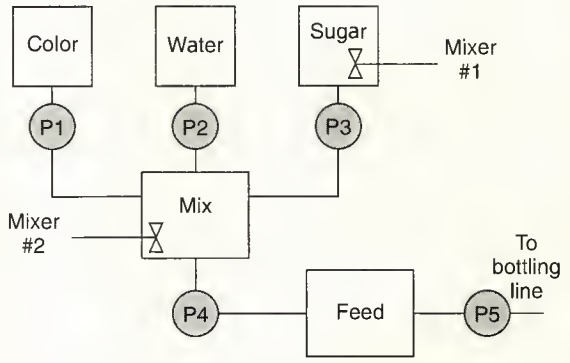


Figure A. Emptying a tank of fluid.

Table A. Configuration.

Tank no.	Tank name	Initial volume	Analog	Maximum volume	Heater	Mix	Pump	Rate	Connect to tank
1	Color	1,000	1	1,000	—	—	1	2	4
2	Water	9,000	2	9,000	—	—	2	20	4
3	Sugar	9,000	3	9,000	—	6	3	20	4
4	Mix	0	4	60	—	7	4	20	5
5	Feed	0	5	100	—	—	5	10	6

Channel	From tank	To tank	Tank name
1	Pump 1 Color	4	Mix
2	Pump 2 Water	4	Mix
3	Pump 3 Sugar	4	Mix
4	Pump 4 Mix	5	Feed
5	Pump 5 Feed	—	

This operating-system table passes into our system and contains pointers to offsets, traps, and interrupts for use in the simulator mode. In this way the student learns to appreciate concepts of I/O mapping, system traps, and interrupts.

Simulator execution

The Execute option requires that the student provide certain qualifications before the plant system will operate.

The student must select the system mode with options of:

- **Wait.** Each instruction executes in response to the student's hitting a key.
- **Slow.** The program executes with one second between each instruction.
- **Run.** The program operates at top speed with no delay between instructions.

When execution begins, the computer screen displays tank levels, status values, flow balance, and also the current instruction being executed. When the program pauses, the

simulator continues updating tank flows and so on while the program waits for the time-out generated by the Pause command to expire. This practice reinforces the notion of rapid processor speed in slow systems.

Instructors can simulate random faults through a control routine. They can model the processor to lose power or data from an I/O channel so that students can test the control program for cohesion and fail-safe operation. In text mode, the system displays a table of fluid volumes and net flow balance only.

The system generates a dynamic display of levels and pump status values and a trace of the current instruction in the program execution in cartoon form.

THIS INNOVATIVE SYSTEM in the teaching of real-time, real-world systems at Penn State is a multifunctional tool for instructional use. Students can configure the system for several different virtual processors due to the software structures that were included in the design for comparisons between assemblers.

The graphics mode is particularly useful in the industrial engineering context because of the visualization of the program execution and obvious fluid flows that the control program enables. Instructors can illustrate techniques and limitations within the language, and students can gain some level of expertise in the software control of real systems. Elementary notions of compilation and operating system concepts are introduced while allowing the student easy access to the simulated plant and a stand-alone processor.

The concepts of process scheduling and sequencing can be taught and examined by requiring students to write code that will cause the model to function safely and realistically while meeting a production schedule.

From the technical viewpoint the system we've described is very interesting. Two methods accomplish the combination of graphics, emulation, and simulation: Modularization of the separate functions into unique program sections and subsequent integration of the parts into a total operating system concept. We defined the files (see Figure 1) so that each part of the real-time project can be modified and then reassembled in seconds for testing. The virtual microprocessor exists only in pointers to instruction blocks with operand fields to match the instruction requirement. In this way, for example, students can access instruction #50 by a BR (DEC Macro II[2]) or a BC (IBM Assembler[3]) mnemonic. The effect is to transfer execution to the listed address. We kept the handling of addressing schemes simple so that the cross-compiler could be generic.

Students can perform interesting sets of experiments using the virtual microprocessor. One experiment introduces the RISC (reduced instruction-set computing) concept by including only a subset of a processor's instructions. For example, though the VAX processor offers over 200 instructions,[4] the instructor can select just 30 or so for student access. Instructors can also set programming assignments that require "missing" instructions, causing the student to learn to "program around" missing elements and to create macro structures and objects.

We plan to add a future module in the system that graphically depicts the processor architecture, showing the interunit dependencies and data transfers that occur during program execution. The system can then be used for teaching computer organization in addition to the real-time, real-world, process dynamics of the current version.

Our system came into being out of the graduate student's need to investigate the interdependencies between control systems, computer architecture, and real-time programming without an industrial laboratory. The programs demonstrate that such instructional tools can be written and realize the goals set forth here.

Acknowledgment

We gratefully acknowledge any incidental use of company trademarks, such as those of IBM and Digital Equipment Corp.

References

1. K.B. Haden, "Simulation of Process Control by Microcomputers to Assist in the Instruction of Assembly Language Programming," master's paper, Penn State Great Valley, Malvern, Penn., Aug. 1989.
2. A. Gill, *Machine Assembly Language Programming of the PDP11*, Prentice Hall, Englewood Cliffs, N.J., 1978.
3. B.I. Burian, *A Simplified Approach to S/370 Assembler Language Programming*, Prentice Hall, 1977.
4. H.M. Levy, and R.H. Eckhouse, Jr., *Computer Programming and Architecture, The VAX-11*, Digital Press, 1980.

Peripheral Hardware and a Hands-On Multitasking Lab

We describe two teaching methods. The first is a set of 15 small, prebuilt hardware assemblies for microprocessor laboratories. The second, a multitasking laboratory-oriented course, lets undergraduate students carry out embedded-control projects, using a low-cost system of microcomputer boards that they wire to prebuilt peripheral boards.

Thomas W. Schultz

Purdue University

While many microprocessor courses either focus on the architecture of the processor or on data processor applications, the hardware modules described here conveniently allow laboratory activities that focus on embedded hardware applications. The multitasking course also described goes even further into real-time systems, as well as teaching group project ideas.

Peripheral hardware for labs

Microprocessors have led to the rapid growth and knowledge of personal computers. Many of us, however, lack the same level of knowledge of embedded control, an application that differs the most from larger computer systems. Most courses stress the processor itself but are weak in the area of hardware interfacing.

Three microprocessor courses at Purdue attempt to fill this need by relying on a set of custom-built hardware assemblies that are easily interfaced to I/O ports. The students plan and carry out the actual connections to the ports and write the interfacing software. Usually this work involves timing and handshaking considerations. These assemblies expose students to the sorts of real devices typically used in small embedded systems without requiring actual construction. Our approach developed over about a decade, superseding several earlier approaches.

Alternate hardware approaches. We first had students use a bus-interface board that plugged directly into the host computer. Initially we used an S-100 system and later a Multibus I system. We provided the students with wire-wrap sockets and wire to install devices such as parallel ports, serial timers, and interrupt handlers. While this exposed students to the devices, so much time was spent wiring and debugging that the actual software and interfacing activity suffered. The weakest students managed only to get lots of debugging experience!

A second approach had the students building desired circuits on small breadboard blocks with interconnections made using solid insulated wires. We called the blocks—purchased for preceding courses—super strips. Assignments have involved making breadboards for a digital-to-analog converter (requiring two or three chips) and a small add-on static memory (involving three chips and many wires).

The students paid for devices they used only once, but even worse, we experienced a wide spread in success rates. Some students progressed smoothly to the rest of the lab, while others debugged wiring the whole period and missed the point of the lab. Their wiring looked like a bowl of spaghetti, and instructors found it almost impossible to help troubleshoot in an acceptable time. One memory expansion lab still uses this approach to give the students an appreciation for the "nonmagic" nature of wiring in memory de-

0272-1732/91/0200-0030$01.00 © 1991 IEEE

vices. The necessary chips are loaned out at the start of the lab, however.

The prebuilt alternative. From these experiences the peripheral module approach developed. We believed students should not have to buy hardware devices that they would need to use only once even though many devices require major development and debugging on the student's part. In year-long design projects they do just that. But, for efficient exposure to as many devices as possible with the accompanying software interfacing, prebuilt modules offer many advantages:

1) **Low cost.** The cost in department funds is insignificant when compared to normal electronic test equipment and computer purchases. Twenty boards can usually be built for less than the cost of one computer, and the modules can be used year after year by students in several courses.
2) **Visibility.** The units are quite robust, but clear plastic tops and open sides keep them from being black boxes. A one-row socket strip takes all computer connections, allowing the students to connect solid wires from computer ports to the devices as desired. Power connections all use color-coded connection posts that can take plugs, clip leads, or single wires.
3) **Available in quantity.** Having multiple units makes it easy to replace a questionable unit in a student's lab (usually proving the hardware is not the problem!). An inventory of identical units makes it easy for the technician to repair units when it is convenient.
4) **Added learning.** Multiple units let us introduce students to new chips and technologies that they would never see otherwise. Documentation packets (usually five to 10 pages specifically written for each peripheral assembly) discuss some design trade-offs not obvious to the casual user. The packets give simple instructions for the beginner and suggest advanced applications for those who might want to accomplish more or build similar devices into a project.

We did not restrict these peripheral modules to any specific processor nor include any microprocessor in the module itself. As indicated, interfacing through parallel I/O ports lets us avoid bus-timing issues. By "making" pulses in software, students become more conscious of the actual signals involved. If the devices were directly interfaced to the processor bus, much of the interface would be transparent and move into the realm of "magic" in the student's mind.

There is no software required to "support" these modules. In essence the students themselves write device "drivers," which include initialization of the liquid crystal display module as well as the I/O ports. The process also involves timing for step pulses or handshaking for the Digitalker board or analog-to-digital converter.

The choice of modules varies from year to year in the three courses. The first course presently uses a lights and switches board, Digitalker, analog interface module, and stepper motor. The second course uses the lights and switches, Digitalker, robotic arms, keypad, and LCD screen. The third course uses the steppers and robotic arms as well as whatever devices the students choose to make easy-to-use interfaces. A picture of an actual lab setup (the hardware of the next section) using one of the peripheral modules is shown in Figure 1. We list the actual peripheral assemblies in the Modules box on the next page.

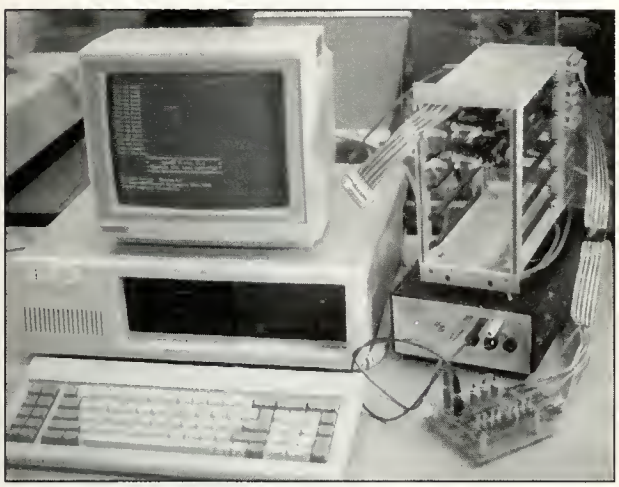


Figure 1. An actual laboratory setup.

Hands-on multitasking lab

While university-level courses on microcomputers may include projects with several I/O devices used at the same time, it is unusual to find such courses using formal multitasking techniques. The variety of external hardware tied to the micro is quite limited as well. Our program is unusual in that its third course in the microprocessor sequence exposes students to both features. Two preceding courses lay foundations in microcomputer architecture, assembly language, high-level language, and programmable peripheral devices. Besides combining the notions of multitasking and distributed control with a variety of hardware, this third course shows the planning required for multitasking systems. The hands-on projects reinforce the multitasking and planning ideas. For about 10 semesters we've had about 15 students participate in each course.

A multitasking operating system. Intel devised a small operating system called DCX (Distributed Control Executive) and a network protocol called Bitbus, around its 8044 microcontroller about six years ago.[1] In addition, we recently read about a small multitasking operating system for the Motorola 68HC11.[2]

continued on p. 80

Peripheral modules

1) One of the first units we developed was a lights and switches board with some debounced inputs. It contains eight SPST (single-pole, single-throw) toggle switches with pull-up resistors, eight LEDs with TTL inverters as drivers (so a logic 1 input turns a light on), and four or five push buttons with debouncing circuitry that produce a logic 1 when pushed. The combination is very useful for most simulations of traffic lights, elevators, and even microwave ovens and washing machines.

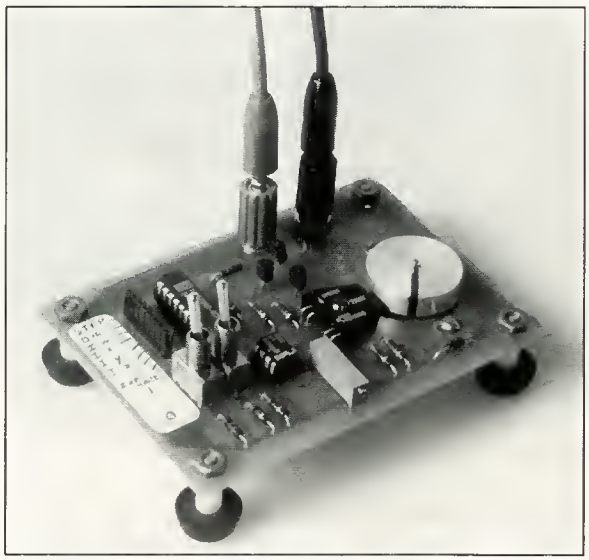


Figure A. A stepper motor.

2) A small stepper motor with a zero-mark position indicator has undergone several revisions (see Figure A). The current units incorporate a PAL (programmable array logic) device that gives four drive modes (full step, half step, and wave as well as direct four-phase unipolar) to show the versatility of such devices. It also includes a reflective emitter/detector assembly aimed at a small aluminum disk with a black mark to indicate a zero position. A comparator converts the light signal from the mark detector into a TTL signal. Again, the goal is to show aspects of technology out in the open.
3) A 4×4 keypad and LCD module shown in Figure B combines input and output. The keypad includes switches to select user-provided matrix scanning or the use of an encoder chip. The LCD is a commercial module (two lines by 20 characters) that supports the alphanumeric character set and eight user-programmable bit-mapped characters. Other than buffering to protect the module, it stands alone.

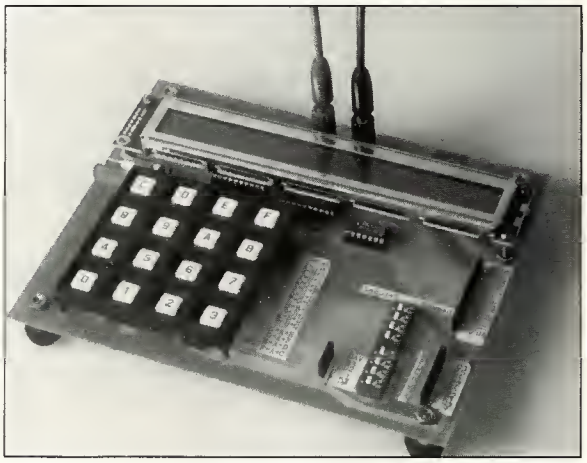


Figure B. Keypad and LCD module.

4) Two modules (under development) illustrate LED displays and multiplexing. One is a set of seven-segment displays with a BCD-to-seven-segment decoder and drivers for the individual digits. A second module will consist of a set of 5×7 LED digits with the flexibility to produce nonnumeric digits. This set will illustrate some brightness issues of multiplexing and will have some sort of latching options as well as the drivers.
5) A speech-synthesizer board with a limited vocabulary (the Digitalker) illustrates high-quality, limited-vocabulary speech synthesis. The device is designed with an 8-bit code for each word and includes a limited set of words, letters, and numbers. The students like the sense of accomplishment they feel when their project talks to them.
6) A simple phonetic/allophone board illustrates artificial speech synthesis. It allows considerable flexibility but illustrates much of the detail required for good sounding speech. A "dictionary" helps students translate to the allophones that must be sent to the board.
7) A "talking" board (Figure C) with an unlimited vocabulary capability uses a text-to-speech algorithm ahead of a phonetic speech synthesizer. It will attempt to say anything one can spell with a microcomputer

using English spelling-to-allophone rules. The user sends the ASCII codes for the characters that make up the words either via an RS-232 serial port or a parallel port. Unfortunately the text-to-speech processor has gone out of production, and the phonetic/allophone unit may have to stand alone in the future.

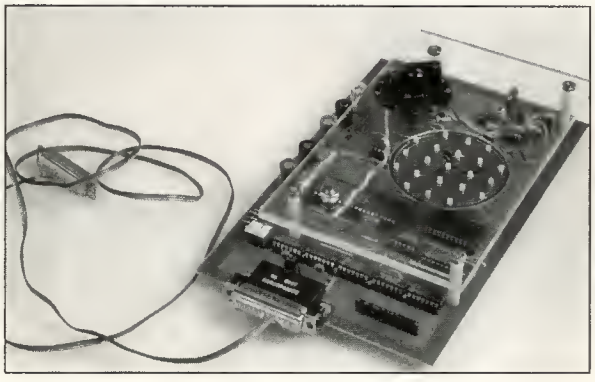


Figure C. A "talking" board.

8) An analog-to-digital board with two types of D/A devices and three A/D devices is the most complex peripheral (Figure D). It is a showcase for all the design options in such interfacing. The analog output can come from an 8-bit D/A or from an frequency-to-voltage converter. The A/D output can come from a 10-bit, successive-approximation converter, a 12-bit integrating converter, or a voltage-to-frequency converter. Students can also produce A/D output in software by feeding an unknown and the D/A voltage (or output) into a comparator.

   In fact, the board offers so many options that some features have yet to see use, but the assembly allows students to try out devices they might incorporate into their senior design projects. Both a digital filter lab and a closed-loop control lab use the A/D and D/A boards at the same time.
9) A programmable music synthesis device (PSG) has been making wolf whistles and whistling bomb sounds for a long time. Recently we obtained and are currently setting up two music keyboards (the mechanical part of what is marketed as an electronic keyboard) as peripherals. Between all the scanning issues for the keyboard and the note frequency and envelope issues, the combination is a challenging project even for groups of students.
10) A DC motor assembly with an on-off driver and a pulse feedback device allows students to develop a closed-loop digital control system. The students usually find the pulse-width drive and the tachometer parts of the project to be much easier to work with than the PID control algorithms, which use only unsigned integer math.
11) A logic-level-to-110-VAC board using commercial optically isolated modules allows exposure to power interfacing and the problems of delays in turn-off of triacs. The assembly includes a ground fault isolation input, several 110-VAC bulbs, and switches (house-wiring devices). A common interconnection area for the high-voltage wiring is central to the board and removed from the 5V logic connections. Common projects include hair-dryer temperature control and athletic scoreboard design.
12) A joystick assembly provides variable frequency pulse outputs to represent the most common method of interfacing joysticks to computers. The purposely slow (25-50 Hz) pulse rate lets students set the timing in a simple program loop or with hardware timers and interrupts. We opened up the original joysticks so the user can see the potentiometers and other inner workings.

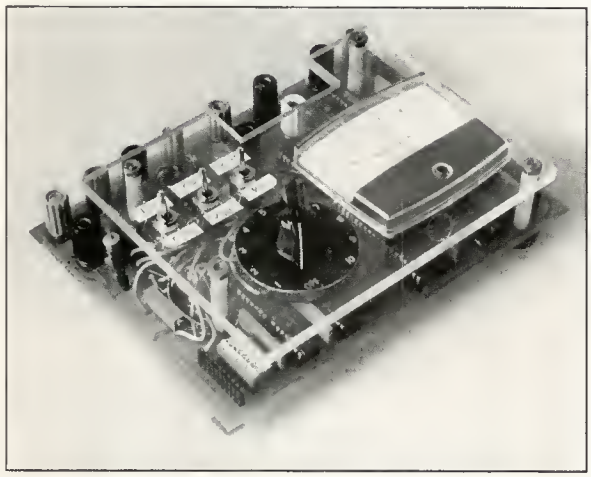


Figure D. An A/D board, our most complex peripheral.

13) Two stepper motors scan a photodetector over a "sky." The peripheral device includes the stepper drivers, a D/A converter, and comparator. The user can choose to use the system as a tracking A/D converter or as a successive approximation A/D converter. Among the

continued on p. 80

Adapting Curriculum Materials for Different Course Sequences

Introductory courses in microprocessing lend themselves to adapation to fit differing levels of student knowledge and skills. Instructors can use a base of 8086 systems and a programming orientation to design a core curriculum for students in both electronic engineering technician and electrical/computer engineering courses.

Douglas V. Hall

Portland State University

The low mathematical content of introductory microprocessor subject matter makes it possible to use very similar course materials in a wide variety of courses. A common core of current microprocessor topics and skills can be adapted for use with high-level electronic engineering technician students or with junior and senior electrical/computer engineering students. The two sequences differ mostly in the rate of teaching, depth of coverage, and emphasis on design techniques.

My experience as an engineer and instructor indicates that it is much more productive to first learn one microprocessor family very thoroughly. From that strong base one can learn others as needed. For most curriculum materials I use the Intel 80xx family of microprocessors because of their widespread use in IBM PC- and PS/2-type personal computers and other systems.

Although now superseded by newer processors, the 8086 is for several reasons still an excellent entry point for students learning about microprocessors. First, relatively inexpensive hardware and software development tools are readily available for 8086-based systems. Second, the advanced features of the newer processors such as the 386 and 486 are not really needed until you discuss multiuser/multitasking systems in later classes. Finally, Intel processors are upwardly compatible, so you don't have to start over when teaching students about newer processors. You just add discussions of pipelining, virtual memory management, and protection.

Since vacuum tubes were my introduction into the world of electronics, I first thought about approaching microprocessor teaching from a hardware direction. However, the more I designed with microprocessors and taught microprocessor classes, the more I became aware that the real essence of a microprocessor is what you can program it to do. Therefore, in each of these course sequences, students first receive a short overview of the computer's operation. Then we introduce them to the 8086 internal architecture, programming model, and instruction set. Contrary to my initial fears, students seem to have little difficulty with the 8086 segments.

Here's how each of the course sequences takes off from this common starting point.

EET sequence

The series of concepts shown in Table 1 has proven to be very successful for a three-term microprocessor sequence in the second year of EET programs. This sequence assumes that the students have previously completed digital courses covering ROM, RAM, buses, and simple address decoding. A previous computer literacy class in which students worked with DOS and a word processor is helpful, but not required.

0272-1732/91/0200-0034$01.00 © 1991 IEEE

Table 1. Electrical engineering technician sequence.

First term	Second term	Third term
Assembly language programming Hardware signals and timing	Interrupts and interrupt processes Digital interfacing and programmable peripheral devices Analog interfacing and industrial control	DMA, DRAMs, and coprocessors C programming CRT hardware and software Disk hardware and software interfacing Serial data communications Networks 386/486 hardware and software

First term. After an introduction to the 8086 programming model and instruction set, the next step is to teach students how to write *structured* assembly language programs for the 8086 microprocessor. We emphasize solving the problem first, writing an algorithm for the solution, and simply translating the algorithm to assembly language.

Students, especially those who have taken previous programming classes, seem to think assembly language is so simple that they can just write a program off the top of their heads. Sometimes it takes the experience of two hours spent debugging a 20-line program to convince them to develop an algorithm first.

For the programming exercises in this section students can use an IBM PC or PS/2 computer, Borland's Turbo Assembler or Microsoft's Macro Assembler, and Borland's Turbo Debugger or Microsoft's Codeview Debugger. Source-level debuggers such as these are very powerful teaching tools, because students can use them to move through a program one step at a time. In doing so, students can observe how registers, variables, and addresses change as a program executes.

Borland and Microsoft each offer generous discount prices to colleges and universities, and to students enrolled in programming classes. With these discounts the assembler and debugger prices are low enough that students often choose to purchase them for home use.

After the students are comfortable with writing and debugging simple assembly language programs, they analyze the signals, timing, and system connections for a simple 8086-based microcomputer. At this point it is important to give some homework assignments that require the students to dig information out of the microcomputer manufacturers' data books. Copies of Intel data books are available free of charge to colleges and universities from Intel's Academic Relations Department.

For the lab exercises in this section we use a board such as the University Research and Development Associates (URDA) SDK-86 shown in Figure 1. One advantage of such a board is that everything is out in the open so students can easily connect a scope or logic analyzer to look at signals. Another advantage of a separate board is that it avoids tying up and potentially disabling the PCs, which are often used for several programming classes.

Students use logic analyzers for observing and making timing measurements on bus signals. Some of the important skills here are writing simple diagnostic loops, choosing a clocking source, and deciding what signal or word to use for a trigger. The concept of pipelining becomes much more real to students when they see the code bytes for prefetched instructions appear on the data bus mixed with data for currently executing instructions. An instructor can also cover the tools and techniques used for systematic troubleshooting of basic microcomputer circuitry at this time.

Second term. A discussion of interrupts and interrupt procedures begins the second term. Instructors can easily add an Intel 8254 programmable timer and an Intel 8259A

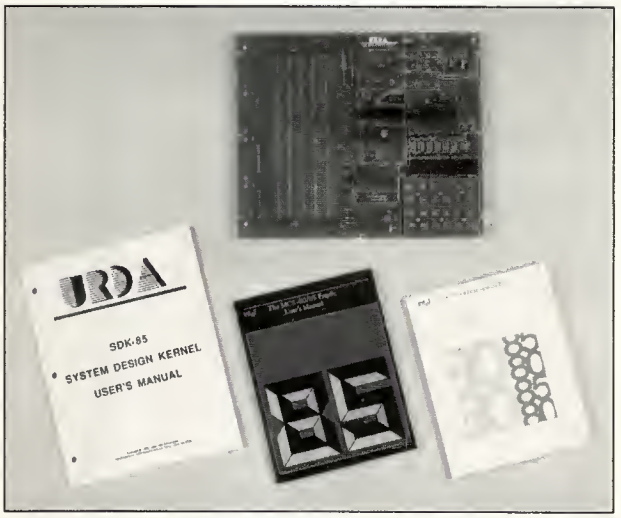


Figure 1. The URDA SDK-86 board and user manuals support laboratory exercises involving signal analysis.

Priority Interrupt Controller (PIC) to the SDK-86 boards so students can conduct a wide variety of interrupt-based exercises. These include interrupt-driven I/O and an interrupt-driven, real-time clock. Students develop these programs on a PC or PS/2 computer and download them to the SDK-86 board over a RS-232C link for execution, debugging, and testing. An alternative to using a board such as the SDK-86 for these experiments is to use a parallel port adapter board with interrupt inputs. The PIO-16/16 from Contec Microelectronics USA is one example of this type of board.

The next major topics in the EET course sequence are digital interfacing and analog interfacing. The parallel port, timer, and PIC on the SDK-86 board are the basis for a wide variety of interfacing exercises. The exercises include keyboards, displays, A/D and D/A converters, speech synthesizer chips, robot arms, and so on.

After learning basic interfacing techniques, the class moves into a discussion of how all the software and hardware pieces are put together to produce a microprocessor-based scale and a simple microprocessor-based process control system. The scale is an excellent example of embedded control, and it leads into a short discussion of the processors specifically designed for embedded control. The factory controller, which uses a time-slice method of servicing eight proportional integral derivative (PID) control loops, is a good example of process control. It also introduces some basic concepts of multitasking, which are developed further in later discussions of the 80386.

Third term. We spend the rest of the available time in the EET sequence on system-level hardware, software, and interfacing. Students learn about the operation of motherboard circuitry on a personal computer such as a PC or PS/2. Included in these discussions are DMA I/O, dynamic RAM (DRAM) refresh controllers, cache systems, math coprocessor operation and programming, and peripheral interface buses.

Industry advisors in my area in the northwestern US point out that most system-level programs are now written with a combination of about 60 percent C and 40 percent assembly language. Therefore, engineering technicians should have some skill with C. Contrary to my initial fears, I discovered that it is very easy to teach C to students who have some knowledge of 8086 assembly language. A little thought reveals why this is the case.

Two of the main stumbling points for students who attempt to learn C before grasping the 8086 assembly language are the inability to work with pointers and to pass parameters to and from functions. A way to smooth the transition is to teach students to use the 8086 indexed addressing modes to work with pointers to arrays in their initial assembly language programming assignments. When C pointers are introduced, it is mostly a matter of students learning a new syntax rather than a new concept.

Demonstrating several examples of the assembly language

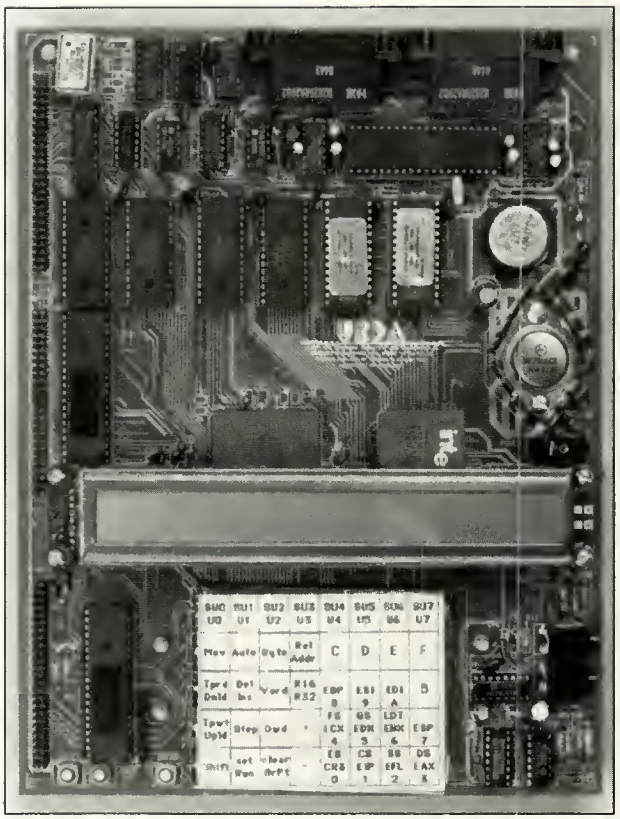


Figure 2. The URDA 80386 µLab board is foundational hardware for student design projects.

equivalents of C pointer declarations and manipulations can also help. With the appropriate command switch, the Borland Tcc compiler or the Microsoft C complier will produce the assembly language equivalent of a C source program.

C functions and parameter passing also hold little more than new syntax for students who have experience in passing parameters to and from 8086 assembly language programs. In fact, the 8086 pointer and function background makes it relatively easy to teach students how to write assembly language modules that can be called from C programs. The key concept here is using the 8086 BP register as a pointer to retrieve parameters pushed on the stack in the C calling program.

For C programming exercises I suggest a system such as Borland's Turbo C++ Integrated Development Environment or Microsoft's Programmer Workbench. These environments allow students to access an editor, a compiler, and a powerful source-level debugger from an on-screen menu. Students can edit a source program, save it to disk with a couple of keystrokes or mouse actions, compile the program with a couple more keystrokes or mouse actions, and run the program with just a couple more.

The debugger included in this environment allows students to display the program source code on the screen and step through program execution one line at a time. Watches placed on important variables allow students to see the values of these variables change as they step through the programs. The point is that these environments speed up the program development process by eliminating the need to type in lengthy commands and providing powerful source-level debugging. The students become highly motivated because they enjoy the graphical user interface.

CRT display hardware operation and interfacing is the next topic. In the exercises for this section students can use C graphics library functions to write simple graphics programs. By leaving exercises open-ended so students can continue as far as they want, considerable competition usually results. In addition to keeping the interest high, this competition gives students some real practice with the C integrated development environment. Subsequent instruction on disk systems and printers will further develop these skills.

For these exercises, students can use DOS or C function calls to open, read, write, and close disk files, and to send files to printers. These system-level programming skills enable engineering technicians to write programs that use a PC to test various products. With these skills they can also often modify existing software to fit a specific need.

Asynchronous and synchronous communications methods and networks constitute the next block of instruction. The instruction emphasizes hardware signals, terminology, and common standards. The area of communications is so large that it is not possible or desirable to try to teach the details of all the current systems. However, a good overview of RS-232 protocols, modems, fiber optics, and networks enables students to read and understand manufacturers' literature.

The final block in the EET topic sequence is an introduction to the 286, 386, and 486 microprocessors. The main emphases are virtual memory, caches, system connections, and signals. If hardware such as the URDA 80386 µLab board shown in Figure 2 is available, students can look at 80386 bus signals by using logic analyzers. This approach refreshes the students' logic analyzer skills.

CE and EE sequence

Table 2 contains a two-term microprocessor design sequence for a computer engineering (CE) or electrical engineering (EE) curriculum. Instructors in the universities I am familiar with teach the first course in the junior year and the second course as a senior elective or design project class. This sequence assumes that the students have completed a digital logic design class that included instruction on basic memory devices and systems.

Table 2. Computer and electrical engineering sequence.

First term	Second term
Assembly language programming	Design project description
Hardware signals and timing	Electronic Design Automation approach to microprocessor system design
Interrupts and interrupt procedures	386 hardware characteristics
Digital interfacing	Hardware cache operation
Analog interfacing	Overview of multitasking operating system needs
DMA, DRAMs, and coprocessors	386 virtual memory and protected-mode operation
CRT hardware and software interfacing	Introduction to selected topics such as RISC, high-performance buses, parallel processors, neural networks, and fuzzy logic
Disk hardware and software interfacing	

First term. The topics are quite similar to those in the EET sequence, but the emphases and timing are different. By the time CE/EE students get to a microprocessor design class, they usually have had an assembly language programming class. Since they at most have to pick up the 8086 instruction set and gain a little practice, students move through the first section of the course on assembly language programming very quickly.

The next major topic is an analysis of 8086 system connections, signals, and timing. Students use data books to help calculate worst-case timing values for a simple microcomputer system and determine whether to insert wait states into memory cycles. Lab exercises for this section require a logic analyzer to determine bus signal sequences and measure timing parameters. For the reasons explained earlier, a separate board such as the URDA SDK-86 board is a good vehicle for these exercises.

As in the EET sequence, the next topics in this CE class are interrupt hardware and interrupt service procedures. It is important to cover both hardware and software interrupts because of their extensive use in systems. Next, the digital and analog interfacing sections are also very similar to those in the EET curriculum, except that we place more emphasis on programmable peripheral devices. We must challenge students to search data books for a device that will help them solve a specified interfacing problem, interpret the data sheet for the device they choose, and write a program to initialize the device as needed for the application. Writing these low-

continued on p. 82

A Futurebus Interface from Off-the-Shelf Parts

As part of the GRIP project we designed a Futurebus interface using standard parts. Our implementation is unusual in its use of fully asynchronous finite-state machines. Based on our experience, we draw some lessons for designers using Futurebus.

Simon L. Peyton Jones

University of Glasgow

Mark S. Hardie

University College London

GRIP (Graph Reduction in Parallel) is a multiprocessor designed particularly to execute programs written in a pure functional language, using graph reduction.[1] As part of its implementation, we designed a bus-based subsystem to interconnect a number of homogeneous modules. For reasons discussed later, we based the communications subsystem on the (then) draft IEEE P896 Futurebus standard. The unusual feature of the Futurebus protocols is that they are entirely asynchronous. They do not use a global clock, and all transactions proceed correctly at the speed of the slowest participating module.

Though previous articles in *IEEE Micro* discussed Futurebus,[2-3] we reintroduce it here. We also introduce GRIP and describe the practical aspects of the design of our Futurebus interface. We pay particular attention to the asynchronous parts, which distinguish it from other bus interfaces. For several reasons, our design does not conform to the current Futurebus standard:

- Primarily, we hoped to achieve high-bandwidth communication between homogeneous boards. We planned to do this at minimum cost in PCB real estate and without using custom VLSI (very large-scale integration) methods. It was clearly not possible to implement the full Futurebus protocols (which are quite elaborate) using this technology. So we chose a small subset that was adequate for our needs.
- In a number of places, strict adherence to the draft standard would have demanded a substantial cost in real estate or speed. Since we did not anticipate interworking with other Futurebus boards, we therefore used a few minor variations of the draft Futurebus standard.[5] We describe each, together with the reasons for adoption.
- We began our system in 1987, basing it on this draft standard. Since then, the standard has evolved significantly into Futurebus+.[6] Nevertheless, the underlying principles of the standard remain unchanged since we designed our interface. The techniques we developed would apply equally to an interface that conforms to the current draft Futurebus+ standard.

The GRIP system

To make sense of the Futurebus interface, we need to say a little about the system into which it fits.

The GRIP multiprocessor consists of a number of conventional Motorola 68020 processors, or processing elements, and unconventional intelligent memory units, or IMUs, connected together using Futurebus. Each PE contains 1 Mbyte of private memory. We packaged the system with four processors and one IMU on each board (see

0272-1732/91/0200-0038$01.00 © 1991 IEEE

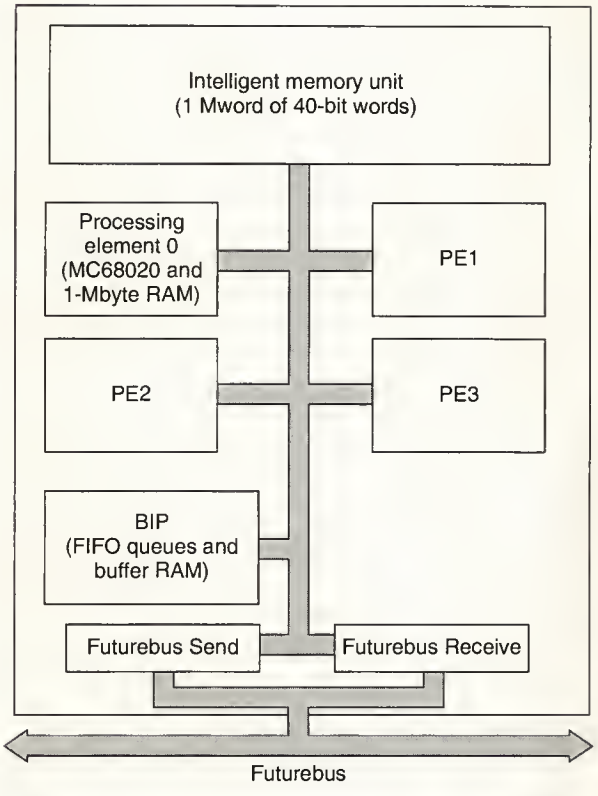


Figure 1. Block diagram of a GRIP board.

Figure 1). The maximum of 20 boards thus allows up to 80 processors and 20 IMUs. Our present system combines 30 processors and 13 IMUs.

An IMU consists of 5 Mbytes of RAM arranged in 40-bit words, together with a microprogrammable data engine. This engine interprets and services incoming requests from processors and dispatches a reply. Because it is programmable, an IMU can support a much richer set of memory operations than the usual memory reads and writes. Our current microcode supports a variety of operations designed to support heap-allocated data structures, such as those required by a Lisp system. Furthermore, the IMU implements in microcode a number of indivisible operations required to support correct synchronization between parallel activities.

Originally, we planned only to build a machine suitable for executing parallel programs written in a functional language.[7] Nevertheless, we did not specialize the architecture for functional languages. Another group currently implements a parallel logic language, Brave, on the same hardware base.

We next concentrate on GRIP's communications system to provide a context for the Futurebus interface.

Packet switching. GRIP uses only packet-switched communication techniques. For example, a processor wishing to communicate with an IMU must send to the IMU a packet that contains the memory request (read, write, or something more sophisticated). The communication system delivers the packet to the IMU. It is then free to deliver other packets while the memory services the operation. Subsequently, as an entirely separate transaction, the IMU sends a reply packet back to the processor. This procedure contrasts with the more common circuit-switched bus protocol. That protocol holds open (and idle) the connection from processor to memory while the memory services the request and replies to the processor. Packet switching offers us far higher bus utilization than circuit switching, at the price of somewhat increased latency.

Communications system architecture. Modules (PEs or IMUs) communicate by sending packets to each other. The sending module writes the packet into a shared block of buffer RAM using a local bus (as shown in Figure 1).

When the receiving module resides on the same board, it reads the packet directly from the shared buffer RAM. Otherwise the Futurebus interface first transmits the packet from the buffer RAM on the sender's board to the buffer on the recipient's board.

The buffer RAM contains a number of fixed-sized packet frames, each of which can hold a packet consisting of an address word and zero or more data words. The address word contains routing information that identifies the destination; the communications system interprets it. The address word must therefore conform to a standard format. The data words can contain arbitrary data.

Each word in a packet is 34-bits long. Of these, 33 are data bits conveyed from source to destination, while the 34th bit indicates the last word in a packet. The hardware does not constrain the 33rd bit to any particular function. Instead, this bit typically distinguishes pointers from nonpointers. At design time, Futurebus provided a 33-bit data highway, though Futurebus+ now provides a data highway of up to 256 bits, together with 8 further "tag" bits.

Each PE or IMU has an associated FIFO (first-in, first-out) queue of packets (or, more accurately, packet addresses) waiting to be processed by that module. Each board contains further FIFO addresses for packets awaiting transmission over the Futurebus, and a stack for free packet frames. We collectively refer to the FIFO queues and buffer RAM, together with some arbitration and sequencing circuitry, as BIP, which is pictured in Figure 2 on the next page. (Somewhat of a misnomer now, BIP originally stood for bus interface processor.)

The processors, the IMU, the Futurebus interface Send machine, and the Futurebus interface Receive machine are each potential BIP masters. When one of them needs to access data in the buffer RAM, it arbitrates for mastership of BIP, accesses the data, and relinquishes mastership. This is,

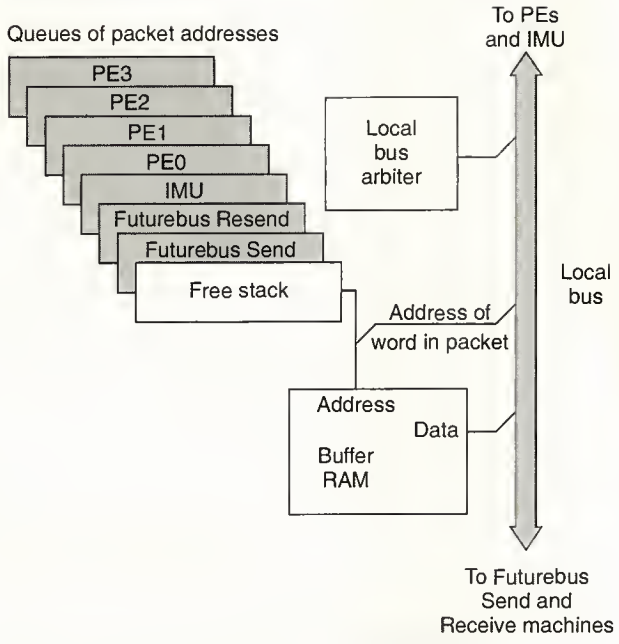


Figure 2. Block diagram of BIP.

of course, an entirely local operation, completely separate from arbitration for Futurebus mastership.

Operation. When a module needs to send a packet to another module, it builds the packet in the buffer RAM. It does so by obtaining mastership of the local bus, claiming a free packet frame and writing data into the packet. The act of writing the address word of the packet causes the packet to be added to the appropriate queue. The Futurebus Send Queue holds packets for remote destinations; otherwise packets enter the Receive Queue of the appropriate local module. It follows that the address word must be written last, when all the data words have been written and the packet is ready for dispatch.

Whenever the Futurebus Send Queue is occupied, the Futurebus interface tries to gain mastership of the Futurebus. When it becomes master, the interface proceeds to transmit all the packets in the Send Queue (which may by then contain several packets) to their appropriate destination boards. As packets are transmitted, their frames return to the free stack.

When the destination board does not have free packet frames, it declines the transfer, and the sending board queues the packet in the Resend Queue for subsequent retransmission. After all packets in the Send Queue have thus been processed, the interface swaps the Send and Resend Queues, and relinquishes mastership of the Futurebus.

When any packets are queued for retransmission, the board will immediately try to become Futurebus master again. However, the fairness mechanism in the Futurebus arbitration protocol ensures that all other boards currently requesting the bus will become master before the retransmission can take place. Retransmission should be rare, so we are not worried about the danger of wasting much bandwidth on it.

As described so far, packets sent from a particular module A to another module B might arrive at B in a different order from that in which A sent them. Reordering could occur if two packets on their way from A to B were both in the Send Queue on A's board. Suppose that B's board refuses the first of them, but meanwhile another packet frame becomes free on B's board. In this case the second packet would be accepted and hence arrive first.

It is highly desirable that the communication system maintains the ordering of messages between any pair of communicating modules. Without this capability, a module that sends multiple packets to another module must attach identification tags to each packet to sort out which reply corresponds to which request. In our system, an extra mechanism suffices to maintain packet ordering. Once a board declines a packet from a particular master, it declines all further packets until the master has completed sending all packets in its Send Queue and relinquished Futurebus mastership.

The P896 Futurebus standard

We sought a high-bandwidth bus that would offer good support for packet switching. Futurebus satisfied these requirements with the following distinctive characteristics:

- The bus drivers and receivers are designed so that a driver can switch all receivers in one bus propagation. Unlike other bus standards, Futurebus doesn't require "settling time."
- The protocol provides a fast, two-edged block transfer mode that we could exploit to transfer packets.
- Arbitration may take place concurrently with data transfer, resulting in only a short hand-over time between bus masterships.

A unifying theme of Futurebus is technology independence. The protocols permit any board, no matter how fast, to interoperate with any other board, no matter how slow. An essential feature of this scheme is the use of open-collector bus drivers. Several drivers may drive a single bus signal simultaneously without damage. When any driver asserts the signal, all receivers see it as asserted. (This is often called a wired-Or connection.)

Futurebus uses uppercase signal names (such as AK*) to refer to the bus signals, while the corresponding lowercase name (ak*) identifies the signal used by the board in question to drive the bus. These two may differ; though ak* may be released, some other board may assert AK*. The asterisk after signal names indicates its active-low state.

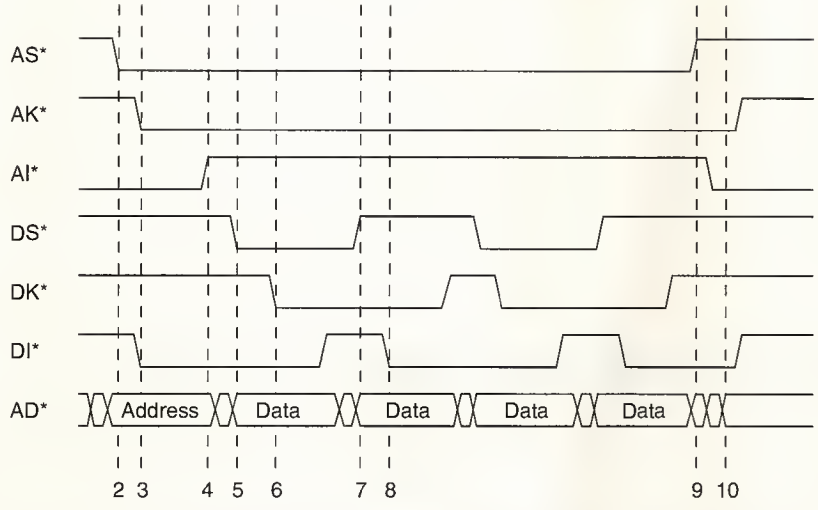


Figure 3. Data transfer protocol. The upper level is released; the lower level is asserted.

Futurebus+ is in the final stages of becoming an IEEE 1990 standard.[6] The full design is quite complicated, supporting broadcast and broadcall, cache management protocols, negotiated noncompelled-mode burst transfer, and so on. Nevertheless, the standard remains true to the core ideas presented in this section. In the following discussion, we indicate wherever the current Futurebus+ draft standard differs from the draft to which we worked.

Data transfer protocol. To understand how data transfers over the bus, look at Figure 3, which shows the interaction between master and slave during the transfer of a 5-word packet. Such a packet transfers in five beats, an address beat to send the first word and four data beats to send the subsequent words. In general, a Futurebus transaction consists of an address beat followed by zero or more data beats. In a conventional system, the address beat conveys a memory address, and the subsequent words convey the data. In our system the address beat merely transfers the first word of the packet. This word contains enough routing information to direct the packet to the correct recipient. But the other bits may or may not contain a memory address, by mutual agreement between the sender and recipient.

The sequence of events during the transfer of a 5-word packet occurs as listed in the Transferring box on p. 84. It should be clear from this sequence why the protocol is independent of technology. Each event causally relates to the preceding one, and no other timing constraints exist. This is a compelled-mode transaction.

The protocol uses two complementary data acknowledge signals DK* and DI*. In this way the protocol can also support a multirecipient broadcast transfer. Since the master always awaits the release of the appropriate acknowledge signal, it will not proceed until all participating recipients have acknowledged (but see the later Wired-Or section).

We've not discussed two important groups of signals so far. The master uses the CM* signals during the address beat to indicate to the slaves what kind of transaction is being initiated. The slaves use the ST* signals to convey status information back to the master. For example, during an address beat, an addressed slave asserts the sl* status signal. The master then registers that at least one slave has been selected—when none are, deadlock results from continuing with the transaction!

Arbitration protocol. The following sketches the Futurebus arbitration protocol, whereby an interface can acquire Futurebus mastership. (Taub[3] offers a much fuller presentation.)

Each board contains an arbiter, which includes a finite-state machine. Each arbiter holds a unique arbitration number, which resolves conflicts when more than one arbiter requests mastership.

Every arbiter moves through the same sequence of states together, at the speed of the slowest arbiter. Three bus signals AP*, AQ*, and AR* achieve this synchronization of state transitions, as follows (see Figure 4):

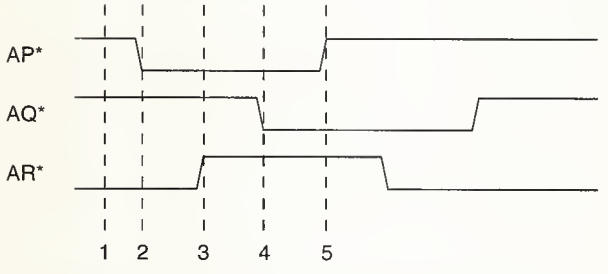


Figure 4. Three-wire synchronization protocol for the arbiter.

1) The arbiter asserts AR* and releases AP* and AQ*.
2) When an arbiter is ready to move to the next state, it asserts ap* and releases ar*.
3) Only when all arbiters are ready to move will all boards release ar*, thus releasing the AR* bus signal. All arbiters

continued on p. 84

When the computer was still personal

Hubert Kirrmann

Asea Brown Boveri

Research Center

CRBC.1

CH-5405 Baden,

Switzerland

On February 1981, the date of the first issue of *IEEE Micro*, the Macintosh didn't exist. Its predecessor, the Lisa, would not be unveiled until two years later. "IBM compatibility" of computers and peripherals referred to the IBM/370 line. The IBM PC was still a well-kept secret.

By that year the pioneering days of personal computers were already gone. Personal computers by Altair, SWTP, Ohio Scientific, Osborne, Cromenco, and other forgotten names filled the junkyards. The surviving garage firms—Tandy/Radio Shack with its TRS III, Apple with its Apple II, and the Commodore with its Pet or VIC-20—relied on the support of large corporations.

At the beginning of the 1980s the computer was truly personal. Everybody had a chance to build a microcomputer from scratch. All the necessary ingredients were present: Z80, 8085, or 6809 microprocessors; 64-Kbit memories; peripheral controllers; Cherry keyboards; Shugard 8-inch or Micropolis 5.25-inch floppy drives; 5.25, 10-Mbyte Winchester drives; and CRT monitors.

Engineers could not be stopped from building their own "personal" computers. It was fun to build one, and even more fun when you could do it in a large firm in which management understood little about microcomputers but thought a modern firm could not do without them.

An engineer could find enough good reasons to build a computer besides "no existing product fits our customers' needs." Low component costs made a do-it-yourself design look very competitive when compared to the hardware costs of Data General or Digital Equipment Corp. However, you had to forget design, documentation, maintenance, and software costs for a while. Many of these systems still live in the industrial world of 1991 since one cannot easily discard a control system like a word processor.

Memory advances

Inexpensive memories were a key factor in the microcomputer boom. The year 1981 marked the beginning of Japanese supremacy in the memory domain. For two years Texas Instruments had promised 64-Kbit dynamic RAMs, but Japan's Fujitsu and NEC delivered them first in quantity at less than $20 apiece. Such product announcements shook the American and European semiconductor industries 10 years ago, but it did not lead to "Sputnik shock" despite numerous political actions that ranged from protectionist bills to government-sponsored projects.

Meanwhile, the US and Europe expended their energies on magnetic bubble memories. Intel, Rockwell, National, and others offered boards with a then whopping 1 Mbyte of memory. Alas, the slow bubbles eventually lost the fight with the 3.5-inch floppy disk. US firms like Mostek retained the lead in static RAMs for only a few more years.

Hot topics

The hottest topic back in 1981 was 16-bit microprocessors like Intel's 8086, Zilog's Z8000, Motorola's 68000, and National's 16000. The 16000 even exhibited a memory management unit at a time when the IBM/370 could address a maximum size of only 16 Mbytes.

National had already paved the way years before with its 16-bit Pace. Texas Instrument's 9900 and DEC's LS-11 were in production. However, engineers did not favor these microprocessors because they found them difficult to handle.

0272-1732/91/0200-0042$01.00 © 1991 IEEE

A 32-bit processor—or at least a processor that pretended to be one—was already around in 1981. Intel's Special System Operation group in Oregon built the iAPX32. The iAPX432 incorporated many innovations in computer architecture. It is probably the most complex processor ever made.

The iAPX432 featured a full-fledged object architecture and a machine-instruction task switch and synchronization procedures—just the opposite of a reduced instruction-set computing machine. Its native language was a forerunner to Ada. Unfortunately, the iAPX432's execution speed was well below expectations. The processor's complexity resulted in disastrous production yields. Intel discontinued the line in 1984.

Computer buses were another hot topic in 1981. Intel's Multibus, the S-100 (IEEE P696), Prolog's STD, Zilog's Z-bus, Texas Instruments' TM990, and DEC's LSI-11 dominated the bus scene. Standardization of microcomputer buses started with the S-100 bus. However, the standard came at a time when the bus market of small PC firms began to disappear. Although the publication of the Multibus (IEEE P796) standard draft occurred in 1980, many still designed their own buses—and ended up with the same compatibility problems.

To anticipate this situation for the 32-bit microprocessors to come, the IEEE's Microprocessor Standard Committee set up the P896 project. Motorola was already working on the Versabus, which would become the VMEbus. Nubus existed in a preliminary version that was completely different from the present synchronous IEEE 1196.

In the first issue of *IEEE Micro*, Andrew Allison disclosed the first draft of P896 (Futurebus). He cautioned readers not to use the article as a design guide. This was wise advice since the only unchanged feature of P896 would be the position of the ground and +5-volt supply pins on the connector.

Unfortunately, continuous improvements delayed Futurebus's standardization until 1987 when nobody needed it anymore. Multibus II, VMEbus, and Nubus filled the same market slot. The high entry price was another reason for the limited success of Futurebus and Multibus II. Most engineers trusted themselves to build a VME or IEEE 796 interface, but they considered Multibus II too complicated and Futurebus frightening. By contrast, the simplicity of the Nubus surely appealed to Macintosh II designers.

Language barriers

Software problems soon confronted computer builders back in 1981. At first, the assembly language of the numerous microprocessors confused people. Many systems used the same mnemonics but in different ways. The IEEE also set up a group to standardize the assembly language, but microprocessor manufacturers never took the resulting P694 language or Common Assembly Language for Microprocessors (CALM) seriously. The manufacturers advocated proprietary development tools to bind customers to themselves. Users turned to high-level languages to allow some software portability.

Pascal was already in use in 1981 (and available on the 68000, 8086, and TI9000). Wirth in Zurich released Modula-2 for PDP 11 and the US Department of Defense published the Ada draft. Interestingly, the existence of structured languages like Pascal did not prevent extensive programming in assembly language nor the emergence of Forth or C. Lisp existed (since 1964!) but outside of the Massachusetts Institute of Technology the industry did not consider it because few electronic engineers understood it.

High-level languages required a development environment, like those offered by Intel, Hewlett-Packard, or Tektronix. Development costs loaded the budget substantially in 1981—up to $100,000. Ada's entry cost was clearly prohibitive and its relation to the US DoD gave it a smell of sulfur in academic circles.

As soon as the programs reached some complexity, they had to manage interrupts and schedule operations. Many engineers began with a small debugging monitor, extended it to a task scheduler, and ended up in the unmanageable complexity of an operating system. Of course, Digital Research's CP/M was available for Z80 processors; however, you needed at least 48 Kbytes of RAM to run it, and it did not support multitasking.

In the past 10 years all products mentioned in the first issue of IEEE Micro have disappeared from the market.

In 1981, Unix already existed for large machines like PDP 11. Microsoft offered Xenix as a Unix substitute for small 8086-based machines. Some real-time kernels were in use, such as Motorola's Versados or Intel's RMS88. Again, the entry price and the licenses were quite high. Many engineers preferred to write their own kernels, and the IEEE set up a standardization committee to help them. Unfortunately, standardization of P855, or MOSI (Microprocessor Operating System Interface), happened at a time one could purchase commercial products for the same task. The interface's success was meager.

Networking also came to the microcomputer in the early 1980s. In December 1980, DEC, Intel, and Xerox

released the "blue spec" for Ethernet, thus ushering in the local area network boom. (The 3-Mbits/s experimental Ethernet had been working since 1975.) The International Standards Organization released the first draft of the Open System Interconnection (Basic Reference Model), which influenced the architecture of all future networks. The microcomputer builders had a hard time grasping these new concepts coming from the telecommunication world.

Successes and shortcomings

In the past 10 years, all products mentioned in the first issue of *IEEE Micro* have disappeared from the market. Some products were successful, such as the 68000. Others, like the iAPX432, were too advanced for their time. Still others, like the NS16000, came too late. The hardware architecture changed little from 1981 microcomputers to the 1991 desktop computers. Components just improved in speed, reliability, and price.

In 1981, few people understood how to write a compiler or design a microcomputer or a backplane bus. In 1991, textbooks detail the microcomputers and buses, yet few people actually implement the design since the products exist. Now one learns to use entire systems rather than individual components, and buys tools rather than creates them.

It seems that standardization missed the goal each time it competed with industrial development. The highly successful IEEE floating-point standard (published in 1981) luckily avoided this hurdle. It is easier to standardize successful products than to produce successful standards.

Issues in the 90s

Progress is an educational issue. Success or failure of a new concept or component depends on how well an engineer understands its costs and promises. Only then the engineer decide what to make, buy, or discard. Complexity is a matter of education. Students coming out of universities tend to use more advanced concepts than deadline-plagued design engineers, not only because students are more foolish but often because they are better acquainted with new concepts.

Today, we have moved to new topics: RISC architectures, signal and graphical processors, expert systems, neural networks, and fuzzy logic. Their complexity will seem trivial once the dust has settled.

Meanwhile, new research waits its discovery. The educational material spread by the IEEE publications—and especially by *IEEE Micro*—has been a catalyst for progress. Let it continue.

Call for Papers

***IEEE Micro* seeks manuscripts for the December 1991 Database Machines special issue.**

Submit six copies of manuscripts by April 1, 1991, to M. Abdelguerfi, Computer Science Department, University of New Orleans, Lakefront, LA 70148, phone: (504) 286-7076; Bitnet MACS@UNO.

Reader Interest Survey
Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 186 Medium 187 High 188

There's a standard hiding out there

Carl Warren

McDonnell Douglas

(714) 896-3311

x. 6-0669

MCI mail: 310-9380

Compmail: c.warren

During the last year we saw a great deal of standards work performed. Some standards reached the end of the process and resulted in useful, promising documents.

For example, the Futurebus standard is now a reality. Companies are starting to consider it and developing products based on the specification. Similarly, the Scalable Coherent Interface (SCI) caught the eye of various chip manufacturers. No doubt, before too much of 1991 gets away from us, *IEEE Micro* will be carrying articles on the hardware and software implementation of SCI.

Falling on deaf ears

Not all standards fared as well, however. IEEE P996, the standardization of the AT bus, has moved extremely slowly. This isn't because work hasn't been done. Indeed Scott Hopkinson, formerly of Emulex Corp. (Costa Mesa, California), and Mike Fong of Chips and Technology Corp. (San Jose) deserve an "Attaboy" for their efforts. Unfortunately, announcements of this standard's work seem to fall on deaf ears—when compared to the sheer volume of interest being garnered by Futurebus, for example.

The interesting P996 attempts to standardize the most popular bus architecture in the industry. The Industry Standard Architecture is so named because some gifted marketing/public relations person used that term when talking to the trade press. And indeed the ISA standard is simply based on numbers. (Depending on which report you read, the number of machines with an ISA backplane reaches as high as 60 million.)

The premise of P996 was to provide a detailed specification for the bus—with or without IBM input. IBM chose not to participate.

A bit of history

The P996 work actually began in the Spring of 1985 at Comdex in Atlanta. Organizers called a meeting of all the key system manufacturers to discuss the possibility of extending the bus to accept 32-bit data. Sounds a bit familiar, doesn't it? I was named the chair of the group, which became known as PC/ET for Personal Computer Extended Technology.

We had two basic goals in mind: Define and standardize the signals for 8- and 16-bit operation, and extend the bus to 32-bit operation. The meetings resulted in several hundred pages of documentation—which would eventually serve as the foundation for the EISA (Extended Industry Standard Architecture) group.

By early 1986 we felt the most promising way to move forward (our mistake) was to take PC/ET to the IEEE for standardization. The result is the P996 work. Unfortunately, the industry lost interest.

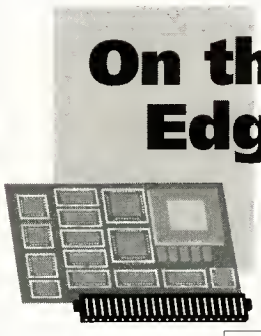
In late 1988 I surveyed the microcomputer industry (70 manufacturers worldwide and 250 independent developers). I asked what their interest was in extending the AT bus to 32 bits. They responded that there was no need to do so and that P996 activities were probably not needed and wasted as far as effort.

Even with this negative response, Compaq was lining up support to develop EISA. You probably know this story. The idea was to extend the AT bus to 32 bits, a function they claim to have achieved. Admittedly, the design is innovative; especially, the two-tiered connector scheme.

However, EISA developers ran into one of the problems with AT bus architectures: There is no clear cut definition of the signals. IBM has never

continued on p. 56

0272/1732/91/0200-0045$01.00 © 1991 IEEE



Evaluating shielded twisted-pair cable

Carl Warren

McDonnell Douglas

(714) 896-3311

x. 6-0669

MCI mail: 310-9380

Compmail: c.warren

When you plan to develop networks consisting of shielded twisted-pair cable, consider that proper operation often dictates that the bus be terminated with an impedance matching the characteristic impedance of the cable. You can determine this characteristic impedance in the lab by using a network analyzer to generate a Smith chart. This chart shows whether the cable is inductive, capacitive, or resistive.

Alternatively, you can determine the characteristics of a particular cable mathematically, given a few of its known variables. The mathematical solutions do not replace the exact results generated by the Smith chart, which is needed for the actual testing of the network. But they may be used as close approximations for selecting this type of cable for use in the testing of the networks.

The following tells you how to develop these calculations using a mathematical tool called Math CAD, from Mathsoft, Inc., (Cambridge, Massachusetts). I'm not giving you a step-by-step tutorial, just indicating what can be done with the right tool.

Constants and variables

To begin, you'll need to determine the parameters for solving the equations of the shielded twisted-pair cable. These include physical constants of permitivity and permeability typical of transmission-line problems and variables peculiar to the cable being used, such as diameters and wire separations. Consider the physical constants listed in Table 1, and the variables we are using in this model as listed in Table 2.

Building the equations

You can perform an initial analysis of a shielded twisted-pair cable using the transmission-line equations shown in Figure 1. The equations apply to this type of cables only and do not yield the same results for twisted-pair cable. The equations provide a method for quickly analyzing the cable's characteristics. They also illustrate the error in assuming twisted-pair and shielded twisted-pair cables behave identically. However, these equations represent merely a textbook solution to the analysis; you must validate them with experimental results.

A tool like Math CAD lets you enter the equations in a form you are familiar with. Notice that I've used the symbolic form to establish the equation. This is the same method you would use when writing the equation on paper. The only difference is that the "paper" is a 24-line-by-80-column screen. You also need to predefine your variables—that is, give them some value that can be used in the equation. You probably already do this when writing the equation on paper.

$$Z_0 = \eta/\pi \bullet \ln\left[(2s/d) \bullet [(D^2 - s^2)/(D^2 + s^2)]\right] \quad \textbf{(a)}$$

$$C_m = \varepsilon_0 \bullet \varepsilon_r \left[(2s/d) \bullet [(D^2 - s^2)/(D^2 + s^2)]\right] \quad \textbf{(b)}$$

$$L_m = (\mu_0 \bullet \mu_r) \bullet \ln\left[(2s/d) \bullet [(D^2 - s^2)/(D^2 + s^2)]\right] \quad \textbf{(c)}$$

$$t_m = \sqrt{(\mu_r \bullet \varepsilon_r)}/c \quad \textbf{(d)}$$

Figure 1. Transmission-line equations: characteristic impedance Z_0 in ohms/m (a); capacitance in F/m (b); inductance in H/m (c); and propagation delay in s/m (d)

0272/1732/91/0200-0046$01.00 ©1991 IEEE

Table 1. Physical constants.

Quantity	Value	Units, symbol	Definition
ε_0	8.842 E – 12	Farad/meter, F/m	Permitivity of free space
μ_0	1.257 E – 6	Henry/meter, H/m	Permeability of free space
c	2.998 E8	Meters/s, m/s	Speed of light
η	254.167	Ohms, Ω	Wave impedance
ε_r	2.2	F/m	Insulation dielectric constant
μ_r	1.0	H/m	Magnetic relative permeability

Table 2. Variables.

Symbol	Units	Definition
AWG	—	Wire gage
d	m	Wire diameter $d = 0.0082525 \bullet 0.890526^{AWG}$
s	m	Wire separation, center-to-center $s = 2 \bullet Th + d$
D	m	Outer conductor inner diameter $D = 2 \bullet s$
Th	m	Wire-insulation thickness

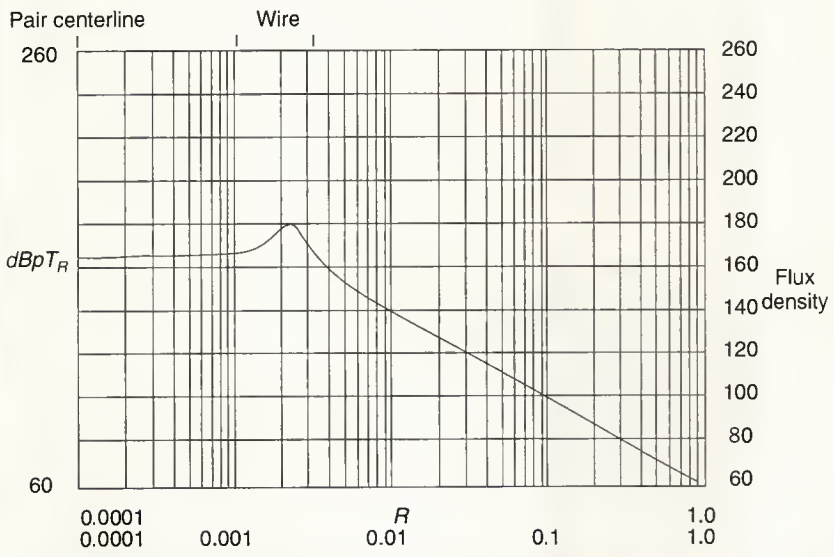


Figure 2. Math CAD graphic with *R* as the termination resistance.

Math CAD isn't limited to just allowing you to enter equations and manipulating an answer. You can also manage a complex analysis and create a graphic display as shown in Figure 2.

This particular graphic describes a cable-generated magnetic field. It shows the effect of two adjacent wires with opposing magnetic forces. The overall flux density is less than the sum of the two fields and is measured in teslas, the unit of magnetic induction.

Without using a tool like Math CAD, creating the graph in Figure 2 would be tedious and probably prone to error. To achieve this graph, I described the equations, established values, and indicated a graph box, choosing the axis names.

In future columns involving calculations, I'll be using Math CAD. If you use this or a similar tool, and would like to share an analysis, contact me for details.

Bibliography

Math CAD Users Guide, Mathsoft, Inc., Cambridge, Mass., 1989.

Electronics Engineer's Handbook, McGraw-Hill Book Company, New York, 1975.

Rudolf F. Graf, *Electronic Databook*, 3rd ed., Tab Books, Inc. Blue Ridge Summit, Penn., 1983.

Standard Dictionary of Electrical and Electronics Terms, 4th ed., ANSI/IEEE Std. 100-1988, IEEE, Inc., New York, 1988.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 192 Medium 193 High 194

The *Paperback* case

Part 3, Misconceptions about functionality

Richard H. Stern

Law Offices of

Richard H. Stern

1300 19th Street NW,

Suite 400

Washington, DC 20036

The court's opinion in *Lotus Development Corp.* v. *Paperback Software International*[1] contains little discussion of the functional aspects of the user interface (input-command vocabulary and syntax) of the Lotus 1-2-3 spreadsheet program. Yet, that should have been a central factor in the legal analysis, because of the consequences to software users of preemption of functional techniques and expedients.

Standardization

The *Paperback* court's failure to recognize the functionality of standardization is a major flaw in its analysis of whether the command structure of Lotus 1-2-3 was functional. The court dismissed the importance of following a de facto standard on two grounds. First, at the most, only following a *de jure* standard could excuse copyright infringement. (A *de jure* standard is one commanded by a government agency or adopted by a standardization organization such as the American National Standards Institute or the IEEE.) Second, standardization is not necessarily in the public interest; the standard might be suboptimal, like the QWERTY keyboard.

The first ground simply misses the point. The *de jure*/de facto standard distinction is a red herring. A de facto standard is simply a widely used convention. Would it be sensible to say that, unless a defendant comic strip artist shows that putting a light bulb over a character's head to show realization of an idea, or using stars to show sensation of pain, is commanded by a standard-making authority, the defendant is guilty of copyright infringement? If a convention is utilitarian or functional, by whatever criterion is judicially accepted for determining that, its use is unprotected by copyright regardless of whether the IEEE or ANSI or a regulatory board has gotten around to declaring it a *de jure* standard.

Perhaps, what the *Paperback* court meant was that it saw a possible conflict between the plaintiff 's statutory right under the copyright law, a *de jure* property right, and the alleged needs of users and competitors to utilize what seemed to be a part of that right. Possibly, the court was saying that in its hierarchy of values, the former (*de jure* property rights) weighed much more heavily, because the copyright laws are intended to reward creativity. But that approach would regard reward as an end in itself, rather than merely

Highlights

Part 1 of this series described the most recent copyright decision in the screen display/user interface field, *Lotus Development Corp.* v. *Paperback Software International*, a decision of great concern to the software industry. Part 1 explored, among other issues, the reason why the defendant copied the Lotus 1-2-3 command tree. Part 2 addressed the court's view of the case, which was that input command structure should be legally protected as a "nonliteral" aspect of the underlying computer program.

Part 3 now discusses the proper role of functionality aspects in a copyright infringement analysis.

0272-1732/91/0200-0048$01.00 © 1991 IEEE

a means of promoting software progress. Or, it would regard the encouragement of user interface creators as the only aspect of software progress deserving consideration when interpreting the statute.

The court's reasoning process suggests that was its unstated premise. Thus, the court supported its conclusion that the 1-2-3 input-command structure was an expression (and therefore protected by copyright) rather than an idea, by pointing to 1-2-3's technical superiority. The court noted that the 1-2-3 command structure became an industry standard because it was so innovative and superior that it swept the field (that is, supplanted VisiCalc).

There is certainly no question but that 1-2-3 was superior to VisiCalc, in terms both of functionality offered to users and quality of user interface. Because 1-2-3 was clearly superior, the court concluded, a rule of law stating that copyright "only protected mundane increments" would "flip copyright on its head" and would be "perverse." Such a legal rule would leave "unprotected as part of the public domain those advancements that are more strikingly innovative," and thus become industry standards. The court felt this would accomplish the opposite of what copyright law should do. Hence, the 1-2-3 user interface should be protected as expression and not considered simply idea. That argument misconceives the issue.

There are several things wrong with the court's logic. First, the court did not address the question whether the most strikingly innovative advancements associated with 1-2-3 were outside the boundaries of the copyright law. Many strikingly innovative advancements are—for example, those in bookkeeping systems, biotechnology, and particle physics. So too, for that matter, are tasty new pies and delightful new perfume scents. Striking innovative advance is a relevant legal test only for whether discoveries are patentable, not for whether writings are copyrightable.

The legal system need not be perverse to deny 75-year copyright monopolies to strikingly innovative technological advances. To do so would more hinder technological progress than advance it. On the other hand, protecting the expressive content of writings with only mundane or no technological value does not help technological progress, but it does not hinder it either. The *Paperback* court simply failed to understand what the Supreme Court was saying in *Baker* v. *Selden*[2] about not confusing copyrights with patents.

The *Paperback* court believed copyright protections should attach to the 1-2-3 user interface to reward its creators for having created a strikingly innovative improvement. At the same time the court expressed a pervasive disdain for standardization if it collided with copyright claims. Presumably, the court thought that its posture would promote software progress in the long run. Viewed in the most favorable light, the court's conclusions reveal a basic tension in applying copyright law to the project of promoting technological progress—whether in user interfaces, other aspects of software, or other technology. The conclusions suggest the existence of basic contradictions in using copyright law as a means for promoting technological progress. There is an inherent tension between a program to reward technological creativity and the traditional copyright doctrine that functionality is not supposed to be protected under copyright law (idea-expression merger).

The *Paperback* court's second ground for dismissing standardization as a material factor is, at best, a non sequitur. The court thought a standard might be suboptimal, as in the case of the QWERTY keyboard. That possibility proves nothing. Standardization is beneficial in the majority of cases, even if and when some standards are suboptimal. Most users would not find it cost effective to learn to use a different keyboard, even though the QWERTY system is suboptimal. For example, probably better gearshift systems exist for cars than the "H" system. On the other hand, who cares? Maybe it could be shown that the world would be improved if green traffic lights meant "stop," and red lights meant "go," or if the green light were placed above the red light. Again, who cares?

The case in favor of standardization of user interfaces, to make life easier for users right now, surely weighs more heavily than the possibility that at some time it may be shown that better results could be achieved under a different user interface standard. (See the box on functionality on the next page.)

A basic misconception of functionality pervades the court's discussion of these points. The court considered that the 1-2-3 command structure was not functional, because it is possible to have a different command set for a spreadsheet. Having more than one way to do something does not mean that some ways will not be better than other ways. Interpreting copyright law to protect the better ways against unauthorized use turns copyrights into patents, without the safeguards and limitations of the patent system. The *Paperback* court simply did not address that point.

But defining functionality in terms of there being any alternative gives copyright owners the right to relegate latecomers to the market to use only less effective alternatives. Those alternatives presumably would be inferior or there would be no controversy. The result would be as if the Supreme Court had told Baker: "There are other ways to keep books besides using Selden's method. Go use one of them."

The *Paperback* court should instead have approached functionality differently. It should have asked and answered the following questions. What is a user interface or input-command structure supposed to accomplish? (Answer: Make it easy for a user to make the program work.) Does that of 1-2-3 do it intrinsically better? (Answer: To

some extent. The user interface of 1-2-3 is certainly easier to use than that of VisiCalc's.) Are there externalities or extrinsic factors associated with the 1-2-3 user interface that make it better in the marketplace—specifically has it become so conventional, or are so many users habituated to it, that it is now somewhat like the QWERTY key-

The functionality of < / >

The treatment of the slash keystroke < / > in the *Paperback* opinion illustrates the court's pervasive wrong-mindedness about the importance of standards and convention. That wrong-mindedness led the court to reach the correct result for the wrong reason when deciding whether < / > should be protected by the copyright in Lotus's 1-2-3 computer program.

The court held that < / > was unprotected, because it was functional. (It is unclear why VisiCalc's prior use of < / > to change from data-entry mode to command mode did not excuse the defendant's use of the same expedient.) But the court found the wrong kind of functionality. Typically, a user interface feature is functional for intrinsic reasons, such as a need to meet requirements caused by the limits of human physiology or by technological constraints.

But sometimes a feature is functional for external or nonintrinsic reasons. Use of such a feature speeds learning and decreases error because the feature is a widely accepted convention, or a so-called de facto standard. Habituation to the QWERTY keyboard is a paradigm for this. Conventions ease the burden on human short-term memory, which is very limited in capacity, by placing the feature in long-term memory instead. See Galitz.[3]

The *Paperback* court chose to rest its finding that use of < / > was not copyright infringement primarily on a conclusion that keyboard-topography constraints functionally dictated the choice of < / >. That conclusion is factually unsupportable. The reason why the defendant used < / >, and the more supportable basis of finding it functional, is clearly that < / > is a recognized convention in the spreadsheet field—indeed, it was what the defendant kept calling a de facto standard. The court disregarded that claim entirely.

The court correctly observed first that a spreadsheet must have a method of invoking the command mode; that function is needed whenever a computer program has the user both making data entry and giving commands. Since we have a limited number of keystrokes, and in different contexts the same keystroke has a data meaning and a command meaning, many types of program need to have a method of invoking a change from data-entry mode to command mode. (Use of function keys for all commands may sometimes resolve the problem.)

The court noted that other possible keys besides < / > can be used for this purpose. Thus, Ashton-Tate's dBase database management program uses < . > ("dot"), the key to the left of < / >, and one approximately as convenient in terms of location. That keystroke cannot be used in 1-2-3, however, because it is needed for data entry of decimal points. A similar problem exists with < , >. The court suggested < ; > as one of a very few possible alternative options. Except for < / >, the court said, "only a few keys are left that can be used, as a practical matter, to invoke the menu command system" of 1-2-3.

Accordingly, the court found idea-expression merger. It said that < / > as an expression merged with the idea of "having a readily available method of invoking the menu command structure." There are so few ways to express this idea besides < / >, the court held, that < / > as an expression merges with idea and therefore cannot be protected by copyright.

Examination of an IBM PC-compatible keyboard suggests that the court overstated the functional dictation of < / > by greatly understating the alternatives. There are many other possible keys, besides < / > and < ; >, such as < ' >, which is adjacent to both of them. The last function key, usually < F10 >, is a perfectly good possibility. So, too, would be < F1 >, if another convention did not suggest reserving it for "Help." There are other keys, as well (for example, the tilde <~>, accent-grave < ` >, and exclamation point < ! >. Although the court said that use of two keys at once, as with <Alternate>, < Control >, or < Shift > plus another key, was objectionable, that is highly questionable. Lotus 1-2-3 frequently uses <+>, < * >, < (>, and other shifted keys. Hence, < Control A > could readily be used, and it is in a very convenient location. We have many other choices available, as well. The court was simply imagining or inventing a difficulty.

Apart from convention and habituation, human factors considerations do not mandate use of < / >. The main reason to use < / > is that it has emerged as a convention for "invoke command mode" in spreadsheet use, and it is useful (functional) to follow conventions. Surely, the explanation for the court's straining to find < / > functionally dictated by the hardware is its complete unwillingness to concede the functional role of convention or de facto standards.

board? (Answer: This is right on the money.)

The court short-circuited those inquiries by asking whether any alternatives existed—never mind how good. Finding that there were alternatives, the court stopped the inquiry at that point. A more thorough factual inquiry would have led to a finding that the 1-2-3 user interface was intrinsically superior to that of VisiCalc, surely, and perhaps approximately equal intrinsically to other alternatives. The inquiry would probably have then led to a conclusion that 1-2-3's user interface is like QWERTY, in its field, and the alternatives are not significantly acceptable to users in the marketplace. To the extent that the court found that 1-2-3's user interface was either intrinsically superior or was "QWERTY-ous," it should have found the user interface functional (and thus unprotected idea rather than protected expression).

Effects on users: Let them eat cake

The tone and flavor of the *Paperback* opinion place it foremost among those recent judicial decisions that raise concerns in the software community. One concern is that the courts apply legal doctrines in ways that interfere with technological advancement. Even worse, what the court said may be perceived as amounting to total disregard of user interests. The *Paperback* court said, in effect, that it was prepared to reward innovation without regard to possible costs to software users or to the effect that such costs would have on software progress. The court's ruling imposes severe costs on users. One is wear and tear on users; another is loss of competition.

The wear-and-tear cost slows software progress. In effect, the court said that users should be compelled to learn new user interfaces (or else, stay with the product that first used that interface). Supposedly, this legal requirement will bring new user interfaces into being, and they might turn out to be superior to those protected by the law. The court felt this policy would promote the progress of science, useful arts, and software.

Reflection suggests, however, that annoying or exasperating users holds back software progress, as explained earlier, by making the public in general less satisfied with using software. That reaction will surely slow the growth of the software market. Enforced competition in user interfaces provides a dubious benefit; it may do more harm than good, by imposing wear-and-tear costs on users. A more useful kind of competition lets companies supply superior functionality to users, in the form of increased task-handling capabilities. These two kinds of competition may well work at cross purposes.

The *Paperback* court's protection of the 1-2-3 command set, as such, is a direct blow against competitive offering of functionality. The command words simply represent the repertory of tasks or functions that the computer program will perform. That in turn defines a market niche or segment at which the computer program will be targeted. Another court held that the selection of tasks that a computer program is to perform is not an act of copyrightable authorship.[4] It said the selection of tasks is idea, not expression, for purposes of determining authorship. That is doubtless right, even if different persons might have different ideas about what the task repertory should be. Hence, any protection of the command set as a set, quite apart from the issue of the tree relationship among commands, is improper.

(Since this analysis of *Paperback* was prepared, the Ninth Circuit held that preparation of an input-command vocabulary for a spreadsheet program was not authorship of copyrightable material. The reason was that command terms are ideas rather than expression.[5])

The effect of protecting a command set, as a set, is to grant the first selector a monopoly on providing users with that set of tasks or functions. Yet, an important part of competition in the software industry involves trying to win away users by offering them an enhanced bundle of functionality: an upwardly compatible superset of the bundle previously available.

Indeed, Lotus 1-2-3 for a time began to lose market share because it did not offer users all of the new functions offered by other spreadsheet vendors (for example, multidimensional or linked spreadsheets). Lotus then brought out new versions of 1-2-3, adding functions that Lotus's competitors had first brought to the market. The same kind of competition occurred with word processors and database managers. It would retard, rather than promote, software progress to declare copying of function repertories (bundles of functions) to be copyright infringement.

In the next and last part of this series I will comment about what is actually going on in the *Paperback* case, and more generally in software cases.

References

1. 740 F. Supp. 37, 15 U.S.P.Q.2d 1577 (D. Mass. 1990).
2. 101 U.S. 99 (1879).
3. W.O. Galitz, *Handbook of Screen Format Design*, 3rd ed., 1988, QED Information Sciences, Wellesley, Mass., p. 17.
4. S.O.S, Inc. *v.* Payday, Inc., 886 F.2d 1081 (9th Cir. 1989).
5. Ashton-Tate Corp. *v.* Ross, 916 F.2d 516 (9th Cir. 1990).

Computer software rentals restricted in US

Richard H. Stern

Legislation passed last December by the US Congress addresses problems involving the rental of copyrighted software and its effect on the sales market for that software.

The Computer Software Rental Act of 1990 attempts to balance the rights of software owners and users. Representative Robert Kastenmeier, chair of the House subcommittee that introduced the law, said the new law gives:

> ... copyright owners of computer programs the right to prohibit the direct or indirect rental, lending, or lease of their computer programs for purposes of direct or indirect commercial advantage. There are, however, three exceptions to this right ... first, nonprofit libraries and nonprofit educational institutions; second, computer programs embodied in a machine or product and which cannot be copied during the ordinary operation or use of the machine or product; and third, computer programs embodied in limited-purpose computers designed for playing video games.

Kastenmeier acknowledged technical assistance from the IEEE Intellectual Property Committee and the IEEE Computer Society's Committee on Public Policy, which corrected some unintended overcoverage in the bill as initially drafted. For clarification, Kastenmeier said the common practice of transferring employer-owned software among employees at the same location or carrying it to other work sites via portable computers is not prohibited by the bill. "The transfer of copies within a single entity, whether nonprofit or for profit, is exempt," he said.

Among other things, the IEEE representatives pointed out to Congress that, as originally drafted, the legislation would outlaw sale-leaseback financing of computers, rental of computerized turret lathes, rental of microcomputers having a ROM BIOS, and rental of any products containing microcontrollers (such as video cameras). Kastenmeier then modified the language of the legislation and accompanied it with a report stating that Congress did not intend to outlaw normal and customary business practices.

Kastenmeier has been responsible for introducing and guiding a considerable body of intellectual property law in Congress. In 1984, he held responsibility for enactment of the Semiconductor Chip Protection Act, which gives a mixture of patent- and copyrightlike protections to chip layouts. In recent years, he had indicated interest in *sui generis* legislation to deal with computer software problems not adequately addressed by US copyright or patent law.

After 32 years in Congress, however, Kastenmeier will not be returning for the 102nd Congress. His unusual sensitivity to technology interests and recognition of the importance of obtaining technological input in legislation affecting technology will be missed. At this time, his replacement on the subcommittee for intellectual property matters is uncertain.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 198 Medium 199 High 200

0272-1732/91/0200-0052$1.00 © 1991 IEEE

Software Report

Optical computing activities

David K. Kahaner

US Office of Naval Research, Far East

kahaner@xroads.cc.u-tokyo.ac.jp

In recent years we've seen the field of optical computing rapidly broaden into investigations of optical analog and/or digital data processing, as well as optical and optoelectronic phenomena and devices for optical computation. Since we accept an optical computer as one in which light is used somewhere, we can study a variety of considerations. Some are fiber optical connections between electronic components, free-space connections, and a computer in which light functions as a mechanism for storage of data, logic, or arithmetic.

Many recent studies of optical computing involved the increasing interest in developing a new parallel computing system capable of processing large amounts of data at high speed. My own interests in the subject center on this potential application.

I wanted to discover how close optical computing is to being of use to the constituency of numerical computing that I represent, and to gain some understanding of the ways digital optical computing and neural computing overlap. In my opinion most scientists engaged in mainstream scientific computation have little knowledge of neural computing and even less of optical computing. Nevertheless, these well-established scientific fields intrigue thousands of researchers, bringing about professional societies, journals, and international meetings. For example, Optical Computing 90, held in Kobe, Japan, last spring, was a major conference on this topic with almost 500 attendees. (A summary and evaluation appears later.)

A number of Japanese see optical computing as an essential direction for computing research. Here are some examples of their comments:

> Electronics is the science of the twentieth century, and optics is the science of the twenty-first.—MITI (Ministry of International Trade and Industry)

> The combination of photons and electrons will create new kinds of systems which we cannot imagine just using an extension of today's technology. For instance, imagine real 3D integration, by which I mean wafer-to-wafer communication, vertically, by light, so that we can make stacks of hundreds of wafers by integration. Once we master optoelectronic integration technology, we can begin to imagine new architectures.—Izuo Hayashi, director of the Optoelectronics Technology Research laboratory, Tsukuba

Comments concerning the research activities in Japan and the US include:

> Compared to the US we have a wider variety of research groups in Japan that are developing devices dedicated to optical computing. In the US, I think there is a wider variety of groups looking for new architectures for optical computing.—Takeshi Kamiya, Department of Electronic Engineering, University of Tokyo

> [AT&T Bell labs researchers] are looking at the short-term target rather than the longer tenn target, and it seems that all their efforts are now concentrated on 1995.

0272/1732/91/0200-0053$1.00 © 1991 IEEE

> But NTT's long-term goal is to establish optical processing technology and to fully exploit massive parallelism by optical means. This will produce new types of optical devices and an optical architecture. We expect that in the long run, research in a broad range of areas will be fruitful. So for now, we consider this to be a basic research phase, not the practical development phase.
>
> The photoelectronic or optoelectronic computer is the direction of the future.—Kenichi Kitayama, NTT Transmission Systems Laboratories.

What is optical computing?

The concept of passing light through lenses to perform computations is not new; I took a course on this in the 1960s. The fundamental idea can be illustrated by noting the difference between it and digital computation. A simple lens essentially performs a two-dimensional Fourier transform of its input in real time for an arbitrarily complicated image. In digital computation the effort increases rapidly with the number of data points or pixels. Using a lens in this way is an entirely analog process, and most of the early research considered computation in analog terms very much like it was described in the days of analog computing.

In recent years developments centered on digital calculations, by using optical devices for logic, memory, or arithmetic. A stumbling block in this research is that one must find optical materials that react nonlinearly to input. Thus far, sufficiently nonlinear materials have not been available, or their nonlinearities are too weak for practical application.

Work also continues on using optics to connect traditional circuits. Optical communication has already made a significant impact in computer communication via optical fibers. It is well known that optical fibers have much lower attenuation during transmission than electrical wires in coaxial cables. In addition, they are more resistant to electromagnetic radiation along their length.

Japan (Hitachi in 1987, Fujitsu in 1988, and NEC in 1989) already uses optical cables as I/O channels. Such channels have data transmission rates up to 9 Mbits per second and may be improved to more than double that. They can be used over much longer distances, up to about 1 kilometer for disk channels, or about eight times as far as electrical channels. J. Goodman (Stanford University), for example, believes that "optical interconnects" in general are promising areas for real products. Further, if we can use these interconnections to connect one chip to another (optical output pads), performance in the 10-gigabit range might be possible.

"Electronics is the science of the 20th century, and optics is the science of the 21st."

Laser beams can cross in arbitrarily complicated ways without losing their individuality, or experiencing crosstalk, at least on larger dimensional scales. Again the Fourier transform provides an excellent example. We obtain each point value of the transform by integrating over all points in the source plane; that a lens can do this easily is, in one sense, the ultimate in parallelism. Some researchers estimate that optics can achieve at least 50 times the parallelism or connectivity of electronic devices.

The Japanese expertise in device technology may enable them to capitalize on this possibility better than others. Kitayama aptly summarized this in describing research at NTT in optical computing:

> Although the applications in the future may be diversified, special-purpose hardware may first come in processing images at data rates which are unobtainable using all electronics. One of the promising schemes would be a combination of optical devices and VLSI. Optical neuro-chips may be a longer term goal.... Practical application of optical hardware still seems to stand at the far end of the time line.

We seem to have four categories of optical computers:

- **Optical analog,** which includes 2D Fourier transform or optical correlators, and optical matrix-vector processors;
- **Optoelectronic**, which does not yet exist, but would be constructed using optical logic gate devices or 2D photo diode arrays; (The main interest in this type of computing device would be to shorten the pulse delay in chips and other logic elements by using optical interconnections.)
- **Optical parallel digital,** which would use the inherent parallelism of optical devices along with digital electronics for flexibility; and
- **Optical neural,** which are specifically designed to implement the massive interconnection requirements of neural networks optically.

General remarks

Optical computing remains a branch of experimental optics, with the usual trappings of physical science, that is,

careful attention to fine detail of setup and analysis. The research is not localized in any one country. Early work by scientists at Bell Labs and other US laboratories is now complemented by comparable work in many other countries.

As long as optical devices utilize free space, the research results are clearly dependent on the planning, creativity, and care of the individual research group rather than on access to technology that is not widely available. Some scientists are beginning to think of optical computing "chips." The Japanese researchers have the advantage of access to the substantial resources and basic technological device infrastructure of large Japanese industrial laboratories. The same comment may apply to work at a very few US labs such as Bell Labs. Thus far, concrete applications are several years away from being useful to the numerical computing community.

We seem to have a healthy competition between major researchers. For example, Huang at Bell Labs is working on optical logic gates based on a principle he calls Seed, but Takeshi Kamiya thinks the NEC approach might have certain advantages such as a capability to amplify optical signals.

Relation to neural computing

A neural computer, or neural network, is a special kind of highly parallel computer with many computing elements, or nodes. The nodes perform simple operations (usually just a matrix-vector product) in a very repetitive manner. Some models of neural computers (which have never been implemented) may call for tens of thousands of nodes, each one of which is connected to all the others. Neural computers compute in the sense that they have streams of input and output bits. They do not require anything resembling ordinary programming; if programming occurs at all, it occurs by dynamically changing the degree to which the individual nodes connect.

An important aspect of a neural network is the high degree of parallelism associated with it. It is natural then that new parallel computers should seek to implement neural networks as an application (but not the only application). Optical computing researchers believe this parallelism can often be implemented best using optical devices rather than traditional wired circuits. Thus the optical computing and neural computing fields, though developed independently, now sometimes come together for their mutual benefit.

While the optics field grows in numbers and import, no certain winner has emerged as the technology.

Today, most applications of neural computing, and in particular those in which optics play a role, are related to image processing, character, target, or voice recognition and similar situations. However, several researchers have demonstrated optical devices that can multiply matrices and solve small systems of linear equations. Others give papers that attempt to apply neural models to more general reasoning situations. As of last spring, neural networks had not been applied to numerical modeling problems, and I saw nothing on any of my visits to suggest that this is likely. Perhaps these models fundamentally differ from what we usually think of as algorithms.

As mentioned above, the fundamental operation of a neural computer is multiplication of an input vector (array) into a matrix with elements called the network "weights." We assume both input array and weight matrix to have nonnegative elements.

Let the matrix elements $W(i, j)$ be associated with a 2D light mask and the input with a light-emitting device array. Let the vector information of the input v be radiated as optical intensity from laser diodes or light-emitting diodes in such a way that $v(j)$ radiates uniformly to the jth column of the matrix W. It should radiate in such a way that $W(i, j)\, v(j)$ is the light intensity on the back, or output, side of the mask at point i, j. We usually describe this as fan-out. Then let the light intensity of the ith row of W converge onto the ith component of a light-receiving device array, also in a uniform way. By superposition, the ith component of the output array is then the inner product of the ith row of W into v. Thus, the matrix-vector product is formed.

The technical issues to be dealt with include developing appropriate fan-out light-emitting beams and a mechanism to permit variation of the values of the components of $W(i, j)$. The matrix-vector multiplier just described is usually part of a "neuron," which takes each output component and returns 1 if it is large enough, 0 otherwise. To do this optically requires optical thresholding.

Optical Computing 90

The 1990 International Topical Meeting on Optical Computing and a related meeting on photonic switching drew participants from many countries. The first day was exclusively devoted to three tutorials, by J. Goodman (Stanford), D. Miller (AT&T Bell Labs), and H. Szu (NRL). A proceedings, in English, is available.

Many of the papers presented at OC90 are variants or extensions of papers that are published in journals such

as *Optoelectronics* or *Applied Optics*. I am not a physicist and cannot evaluate such aspects of this work.

Yoshiki Ichioka and colleagues (Osaka University) have been concentrating on building optical devices that will perform fundamental logical functions, And, Or, Nor, etc. The group created several new ways to implement some logic devices. Their Opals computer design uses these techniques. This work has been going on since at least 1983, and Opals is well known outside of Japan. Since implementation requires development of several new kinds of devices, this research remains several years away from practical application.

Goodman and colleagues from Stanford described a simulated 64-node, shared-memory multiprocessing system, the IBM RP3, with optical interconnects. Similarly, staff (Matsumoto, Sakano, Noguchi, Swabe) at NTT Transmission Systems Lab described a system composed of 36 T800 transputers. This system runs a parallel processing system with optical connections in "free space," that is, in a box rather than in a chip. The technique looks promising but is also at a very early stage.

One of the most interesting papers described Mitsubishi's Optical Neural Neurocomputer. While many research activities, both in academia and industry, seek to develop optical chips, Mitsubishi's looks very impressive to me. Principal researchers Kazuo Kyuma and J. Ohta actually fabricated a device that they claim can be mass produced.

A important aspect of this chip is its two layers—other designs use three. The researchers use a sensitivity-variable photodiode. Further, the chip permits dynamic alteration in the interconnection weights between input and output.

This last feature is essential in neural networks for "learning," the adjustment of interconnection weights to obtain specified output for given input. Many other projects lack this capability to vary the weights.

The Mitsubishi group estimates that it can build this chip to contain about 1,000 neurons in a 1-cm square with acceptable power requirements (less than one watt), large dynamic range (20 db) for the weights, and low optical crosstalk. Although light beams do not significantly interact at large scales, at chip-level scales they do, and this issue must be addressed. The researchers estimate that such a chip can perform more than 1 teraconnection per second, 10^{13}.

Conclusion

Optical computing appears to be a research area that has already provided a few practical applications. Its real potential for general computation lies several years away. There are many steps from discussion of And gates to Fortran compilers. Nevertheless, this seems like a research topic with high payoff potential and only modest risk (cost). The Japanese government has set up an optoelectronics laboratory at Tsukuba. It would be natural for them to enlarge the scope of its research into optical computing, perhaps in conjunction with the activities that are present at the industrial labs.

[Editorial Board member David Kahaner traveled in Japan on assignment with the US Office of Naval Research, Far East. His comments are his own; they do not express any official policy.]

Micro Standards

continued from p. 45

fully released that level of documentation. Most designers have analyzed existing implementations and developed an architecture. The key signals for memory reads and writes, plus various processor controls, have sufficient margin that most boards work in most cases. Consequently, you end up with a wide range of performance.

Obviously, a strong case can be made for standardization. The P996 group continues to work to this end. The standard, which recently went to ballot, has had favorable response with some comments, according to Mike Fong.

Setting the stage

Even though the P996 effort appears to be of little interest, it does have the possibility of setting the stage for future ISA and EISA bus work. With a codified specification to work with, designers can reach some middle ground of technical consensus—a problem that the EISA designers have yet to face up to.

To guarantee long life and further developments to their work, no doubt the architects of EISA will have to pay much closer attention than they have in the past to the way IEEE and other widely accepted standards are defined. This may prove to be a big pill for them to swallow—and EISA can die the gentle death it deserves.

If you're interested in more information about the IEEE P996 Draft Standard (Version 2.01), contact Scott Hopkinson, Chair, IEEE P996, PO Box 3237, Long Beach, CA 90803; or phone him at (213) 934-1235.

New Products

Send announcements of new microcomputer and microprocessor products to Managing Editor, IEEE Micro, PO Box 3014, Los Alamitos, CA 90720-1264.

Joe Hootman

University of North Dakota

Signal processing components

■ DSP built for voice and data

Based on Motorola's 56000 architecture, the 16-bit 5616 DSP offers optimized performance for voice and data applications. According to the company, the 5616 processes 40 million instructions per second (MIPS) with a 25-ns instruction cycle time. Auto return interrupts permit the DSP to stop, handle a data interruption, and return to its previous state. *Motorola.*

Reader Service No. 10

Boards, chips, and components

■ Skewing time

Six low-skew clock drivers assist designers in minimizing timing problems and increasing system performance. These IC clock drivers—74AC11204, 74ACT11208, 74AC11208, SN74AS303, SN74AS304, and SN74AS305—distribute generated clock signals. The devices provide a tight propagation delay distribution by establishing an output-to-output, pin-to-pin, and package-to-package skew. *Texas Instruments.*

Reader Service No. 11

■ Taking a RISC

A one-chip microprocessor brings reduced instruction-set computing (RISC) performance to embedded systems. The LR3300, based on Mips Computer Systems' 32-bit RISC architecture, incorporates 8 Kbytes of instruction cache, 1 Kbyte of data cache, three memory modules, and an integrated dynamic RAM controller.

In addition, a user can scale power consumption directly with frequency or reduce it for battery-powered applications. *LSI Logic; $99.95 each (1,000s).*

Reader Service No. 12

■ Chip matches 68030 performance

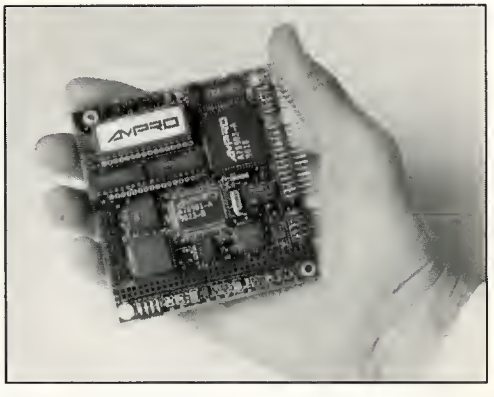
A version of Motorola's 68030 microprocessor uses dynamic memories to attain performance levels comparable to RISC processors. The company states that the 68EC030 chip matches the 68030's performance with a throughput of up to 9.2 VAX-equivalent MIPS when clocked at 40 MHz. The EC030 also features object-code compatibility with 680xx microprocessors; burst-mode bus interface; separate on-chip caches; and a 4-Gbyte addressing range. *Motorola; $75 each (1,000s).*

Reader Service No. 13

■ Hand-held CPU module

Measuring 3.6 × 3.8 × 0.6 inches, the Coremodule/XT is a fully configured, PC-compatible CPU module for embedded applications. It includes memory, solid-state disk, serial and parallel I/O ports, real-time clock, and keyboard and speaker interfaces. Coremodule comes equipped with a CMOS-enhanced, 9.82-MHz microprocessor. *Ampro; $130 OEM (1,000s), available four weeks ARO.*

Reader Service No. 14



Ampro CoremoduleIXT

■ Controller for VGA circuits

A video graphics array controller allows OEMs to build a complete VGA circuit for IBM PC and Micro Channel bus systems. The AVGA1 includes video D/A converters, clock synthesizers, monitor detection, and system bus and feature connector interface logic. A 128-pin, plastic quad flat-pack controller provides access to video memory through the use of waitless write cycles and zero-wait-state bus requests. *Acumos; $22.50 each OEM (10,000s).*

Reader Service No. 15

Memory chips

■ SRAM solutions

Two 17-, 20-, and 25-ns static RAMs with byte-write capability and noise minimization features join Motorola's 1-µm CMOS fast SRAM family. The MCM62995 latched RAM provides memory capabilities for asynchronous applications. It also offers a 128-Kbyte cache solution for a R3000 microprocessor operating at 33 MHz. The synchronous MCM62990 provides general cache solutions using clock speeds up to 40 MHz. *Motorola; from $39 each (1,000s).*

Reader Service No. 16

■ DRAM speed

A 1-Mbit, CMOS DRAM allows for DMA with 16- and 20-MHz microprocessors. According to the company, the AAA1M300 series operates at 53 ns, consumes 400 milliwatts with 5-volts Vcc, and achieves read/write cycle times of 100 ns. Consequently, designers can create zero-wait-state systems using 20-MHz microprocessors without conducting complicated interleaving or caching schemes. *NMB Technologies; $5 each (10,000s).*

Reader Service No. 17

■ Retain memory for 10 years

The GR281-4 nonvolatile RAM chip features a lithium power cell that retains memory contents for up to 10 years. Fabricated from silicon-gate, CMOS technology, the GR281-4 is compatible with a normal SRAM. Its applications include high-speed data collection and use with microprocessors. *Greenwich Instruments; $21 each (1,000s), stock to two weeks.*

Reader Service No. 18

Image processing

■ One-slot image boards

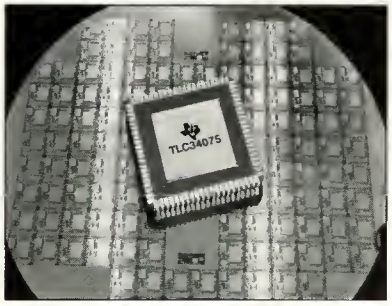
Two image processing boards each occupy only one slot in an IBM PC. The DT2878 Advanced Processor's true fast Fourier transform capability allows data analysis from partial or whole frames in the frequency domain. Users can process data in either floating-point or fixed-point formats. The DT2868 High Speed Frame Processor uses a pipelined architecture and performs image processing faster than older DT processors. *Data Translation; DT2878, from $4,495; DT2868, $1,995.*

Reader Service No. 19

■ VIP treatment

The TLC34075 Video Interface Palette (VIP) combines a programmable color palette and video circuitry on one chip. The chip provides VGA pass-through and integrates pixel data path control logic, video overlay, and timing logic and interface to video RAM. It also supports a range of resolutions and colors. According to the company, VIP incorporates functions that previously required up to 30 chips to handle. *Texas Instruments; samples available.*

Reader Service No. 20



Texas Instruments TLC34075 Video Interface Palette

■ Real-time video software

IP Lab-PPG software, used in conjunction with the Perceptics Corp.'s Pixelgrabber/Pixelpipe board set, permits real-time video display and processes at 30 frames per second. IP Lab-PPG provides user-selectable options including real-time averaging, multiframe integration, background subtraction, and area of interest masking. It also supports point functions, linear and morphological filtering, FFT capability, and statistical and binary operations. The software requires a Macintosh II computer with a minimum of 4 Mbytes of RAM, and an 8-bit video card. *Signal Analytics Corp.; $749.*

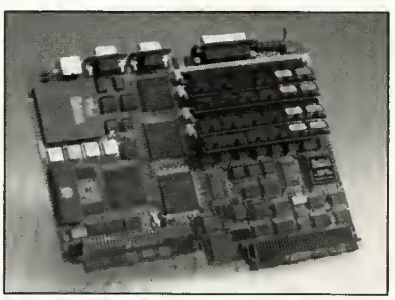
Reader Service No. 21

Computer systems

■ CPU includes 486 processor

An embedded system subassembly, the EPC-5 CPU combines a 25- or 33-MHz 486 microprocessor, a maximum of 16 Mbytes of memory, two serial ports, and a VGA graphics controller. It also contains RISC-style instruction-set optimization; 8 Kbytes of on-chip cache; an eight-layer board with 4-Mbit DRAMs; and dual-bus architecture. According to the company, the CPU also operates at more than 27 MIPS. *Radisys; from $7,495.*

Reader Service No. 22



Radisys EPC-5 CPU

■ Portable 486 PC

Measuring 16 × 9 × 7.5 inches, the Regal 486-ISA personal computer features a 100-Mbyte hard disk; 1.4-Mbyte,

3.5-inch floppy-disk drive; 4 Mbytes of RAM; and an 8-Kbyte RAM cache. Also included is a 11-inch, high-contrast gas-plasma display that tilts for user viewing. *Micro Express; $3,999.*

Reader Service No. 23

Micro Express Regal 486-ISA personal computer

■ Desktop speed and security

The Bravo 486/25 desktop computer includes 2 Mbytes of memory, password security, and an integrated board with 16 Mbytes of memory expansion. The system also offers 8 Kbytes of integrated cache, a numeric processor for the 486 chip, and support for four drive bays. The computer uses surface-mount technology and a custom ASIC design. Four models are available. *AST Research; from $3,995.*

Reader Service No. 24

■ RISC processor executes at 16 MIPS

Version 960 SA is a 32-bit, embedded RISC processor for I/O processing and imaging applications. According to the company, it operates at 16-MIPS burst execution and at 5-MIPS sustained execution. Version 960 includes a floating-point unit for math processing. Both models contain 16-bit external buses. *Intel; from $19.20 each (1,000s).*

Reader Service No. 25

Display products

■ Four slim EL monitors

Four electroluminescent, flat-panel display monitors offer compatibility with IBM PCs, Macintosh IIs, and Digital VAXstations and DECstations. All of the 13- to 19-inch monitors are less than three inches thick. The WS-XT/AT monitor comes with a 1214 controller card that supports Microsoft Windows 3.0 and IBM 8514/A software in SVGA, VGA, and EGA modes. The WS-Mac monitor includes a 1214-Mac Nubus controller card. *Planar.*

Reader Service No. 26

■ Multimedia projections

The Magnabyte 5090-120 multisynchronous computer projection panel presents a variety of features for multimedia purposes. With a cable change, the system becomes compatible with VGA, Macintosh II, EGA, CGA, MCGA, monochrone, Hercules, and Mac SE computers. The color liquid crystal display panel contains 47 preassigned magenta palettes, which each hold a combination of 16 color shades or patterns. The user can change computers without making setting adjustments each time. A wireless remote is also available. *Telex Communications; $1,795.*

Reader Service No. 27

■ Glass screen for humid areas

The Crystal Clear Touch Screen uses nonreflective, clear glass for improved optics and resistance to scratches. The company states that its screen permits more than 95-percent light transmission with no dispersion or diffusion. The hermetically sealed glass surface allows the screen to function properly in highly humid environments. The screens fit most sizes of CRT and flat-panel displays. *Interaction Systems.*

Reader Service No. 28

Interaction Systems Crystal Clear Touch Screen.

Miscellanea

■ Software for C++

The graphical design editor in the Object-Oriented Structured Design (OOSD)/C++ software automates OOSD notation for C++. Features include drawing rules, a reuse library and browser, data modeling editors, document preparation, and version control. OOSD/C++ will use the X Windows system on a number of workstations. *Interactive Development Environments; from $8,500, available in phases during 1991.*

Reader Service No. 29

■ Computer theft prevention

Once installed inside a computer, the PC Screamer detects unauthorized tampering or theft and blasts a siren. It continues to sound during the removal attempt. The alarm resets 30 seconds after the last attempt to move the computer. Measuring 3-3/4 × 2-3/8 × 1 inches, the PC Screamer fits inside the cases of a variety of computers. *Vantage Point Technologies.*

Reader Service No. 30

■ Micro blasting

Technicians can remove conformal coating from PC boards and conduct similar small-scale operations with the Micro Blaster tool. The tool's nozzle propells fine, micrometer-sized powder and air at high pressure onto the working surface. Operators can focus the abrasive at the targeted spot by using a pencillike handpiece. A range of nozzle sizes and abrasives are available. *Comco.*

Reader Service No. 31

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 189 Medium 190 High 191

Product Summary

Joe Hootman
University of North Dakota

Manufacturer	Model	Comments	R.S.#
CAD tools			
Chronology	Timing Designer software	Front-end package allows electronic engineers to draw and analyze timing diagrams for digital circuits. Mouse-driven program performs timing analysis calculations and holds a database for storing generated data. Software runs under Microsoft Windows 3.0. $1,495.	80
Motorola	H4C Series gate arrays	Line of CMOS gate arrays contains up to 318,000 available gates. According to the company, the 0.7-µm series achieves higher density and performance by shrinking array features and using a reduced gate-oxide thickness. The H4C Series operates at 180-picosecond typical gate speed and offers embedded boundary scan logic. The line is scheduled for introduction in April 1991.	81
Meta-Software	Hspice H9007 simulator	Enhanced version of the optimizing analog circuit simulator offers a Graphical Simulation Interface, simulation visualization environment, and a new mixed-signal interface. H9007 also features new user-defined models, expanded behavioral modeling options, and an improved transmission line model for board/hybrid and LSI applications. From $3,000.	82
Accel Technologies	Tango-Schematic Series II software	Version 1.20 supports Pspice and P-CAD net list formats. Added features include user-defined keyboard macros, autopanning, wild card searching, and a snap-to-pin capability. Tango-Schematic also contains component libraries of 10,500 parts. $495.	83
VLSI Technology	VGT and VSC libraries	VGT350 gate-array and VSC350 and VSC370 standard-cell libraries offer 1-µm performance for high-end workstations. The company says the VGT350 has a typical gate delay of 260 picoseconds, and the VSC370 holds double the cell design density of the existing VSC300 library. $5,000 each (commercial), $15,000 each (military).	84
Mentor Graphics	GDT Designer tools	New toolset allows ASIC vendors and IC designers to port libraries to new IC processes, automatically characterize cell libraries, and capture layout procedures and parameters graphically. GDT Designer includes a mixed-signal, multilevel simulator; placing and routing; and rule checker. Available for Sun and Hewlett-Packard/Apollo platforms. $90,000.	85
Source III	Xmenf utility	Vtran module option translates Mentor Force files to stimulus formats compatible with many logic simulators. Xmenf extends the capabilities of Vtran so it can work with high-level language constructs such as Do macros, loops, and variables. Available on Sun, Apollo, and Intergraph platforms. $2,495 (single license).	86

Looking back

continued from p. 9

The *Micro* vision

There is a real need in the IEEE for a journal devoted to system design. This would include hardware, software, design trade-offs, ease-of-use considerations, application design and development technology, and so forth.... The primary factor which made our *Transaction on Software Engineering* and *PAMI* successful was the availability of a capable, dedicated champion for the publication who was willing to assume the responsibility of editing it.—T. H. Bonn, September 10, 1979

The other approach to the new publication is from the readability level of material angle. This approach has not, to my knowledge, received much attention, and I would like to discuss it further here. My basic intent is to provide a publication which covers the technical spectrum as does *Computer*, but whose basic format has unique aspects which orient it primarily towards practicing engineers, programmers, and computer scientists. Such a publication would have a unique character in the publications menu of the CS and would enable us to emphasize service to that part of the CS which contains over 40 percent of our members...."—Roy L. Russo, November 2, 1979

The commonality of [other members of the Governing Board] concerns with mine, together with ideas developed in meetings I had with Dick Merwin and Rex Rice, led me to place the motion,

> IEEE Computer CS initiate publication of a new journal dealing with microprocessors, microcomputers, and systems applications starting in January 1981. Such journal will cover both hardware and software aspects and can include program listings. Whatever activities required to meet the above goal shall be started immediately. The CS will cooperate with other IEEE groups and societies in developing this publication. Ten thousand dollar initial start-up funding will be provided to support planning.

To genuinely serve all of the engineers in the IEEE and the CS, I believe a suitable balance must be found between esoteric mathematical presentations aimed at the half dozen other experts in the topic (and I have been guilty of those myself), and the articles directed toward a more typical bench engineer or junior programmer. It is the latter category of IEEE members who have been ill served by the IEEE *Transactions*, which as a rule tend to contain articles written by the creators of some new thing-a-ma-bob rather than the users of them. Further, the style and content of some of the *Transactions* has been devastated by the idiosyncrasies of certain editors. The style used so effectively in *Computer* should serve as a good initial model for *Micro*.

My hunch is that there will be no shortage of prospective papers for *Micro*... the high degree of cooperation I've experienced in the CS, and the good will which exists for the IEEE should ease the task of developing a good base of papers."—Robert Stewart, March 7, 1980

Problem areas: We are entering a field with lots of competition from other [non-IEEE] publications for papers and readers. The establishment of reliable sources of quality papers may prove very difficult. We need CS TC and conference support in this area. Special joint issues with other TAB S/G/C should be encouraged....—Richard Merwin, March 17, 1980

be fair to say that the publishing business is one of the true high-technology businesses around. The cost of publishing *Micro* has decreased because of the use of the computer and the progressive printing contractors. For example, all manuscripts arrive on disk ready for copy editing, and the edited magazine arrives at the printer ready to be output on film. Thus, the staff can spend more time and energy working with the author in clarifying the text and in formatting the magazine.

Being the EIC of *Micro* was one of the most satisfying professional experiences that I have had in years. The people that I have worked with have been most professional. I have met some very interesting people and have very much enjoyed working with them.

Dedicated tools

continued from p. 17

users, but it confuses beginners and leads to a lot of errors when changing over from one processor to another.

CALM uses a simple set of mnemonics, makes the size of operands explicit, and holds addressing modes that read clearly.[21] One book, which unfortunately is too expensive for students, documents seven processors consistently.[22] We needed to make very few changes to assembler notion as more complex processors became available. Z80 designers changed the addressing mode notation toward CALM for the Z8000; Motorola made a similar modification from the 68000 to 68020.

For example, the 68020 post-indexed indirect mode, as shown in Figure 2, is written in CALM:

```
Move.16  {[A1]+Disp1} + A16^[D0]* + Disp2, D3
```

This is easy to decode: The content of A1 is added to the operand Disp1. This result is considered an address that points to a memory address (indirect addressing), the content of which adds to a value depending on D0. The 16-bit content of D0 is sign-extended and multiplied by 2. The asterisk signifies automultiplication, which occurs by 2 here since we move 2 bytes (Move.16). The Disp2 operand adds before fetching a 16-bit word at the resulting address, and saving it in D3. The prefix A16^ could be inserted in front of the two Disp operands, but this is unnecessary when no choice exists or when the choice can be left to the assembler.

The Motorola notation for this is

```
Move.W  ([Disp1, A1], D0.W*2, Disp2), D3
```

An important point about CALM is that it avoids the confusion of the size of a word, a half word, or a long word. For noting these lengths of 8-, 16-, and 32-bit words, students have voted in favor of .8, .16, .32 instead of the typical .B, .W, .L suffix used by Motorola.

In addition, CALM distinguishes between arithmetic and logic word extensions. One difficulty of the 680xx is knowing when to consider 16-bit operands as addresses and sign-extended functions. A16^ is a space specifier, which means the following operand is an arithmetic number that can be sign-extended. Many programmer errors occur as a result of the writing of Motorola's code:

POP.W	D0	Pop 16 bits into a data register
POPA.W	A0	Pop 16 bits and sign-extend it into an address register
POPM.W	D0 … D3, A2	Pop five 16-bit words and sign-extend them all before loading the aforementioned registers

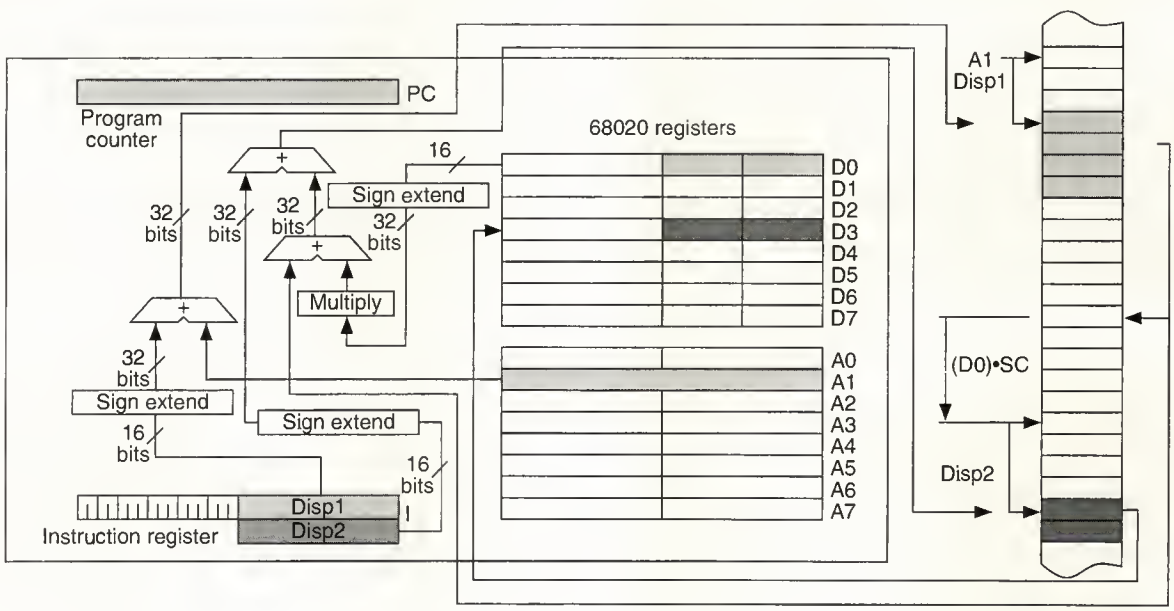


Figure 2. Post-indexed, indirect 68020 addressing mode.

CALM forces the user to understand the processor and emphasizes the anomaly of the sign extension in the multiple Pop instruction:

```
POP.16   D0
POP.A16 A0
POPM.A16          D0 ... D3|A2
```

CALM syntax[23] and comments[24] are available upon request. A firm founded by a former student offers low-cost cross-assemblers for more than 20 processors on PC and compatibles, Atari, and Smaky. DIN, the German standards organization, standardized CALM starting in 1985. The International Electronics Commission is in the process of accepting CALM as an international standard.

At this point, the IEEE 694 group did not follow the notation of CALM, because CALM requires a few extra keystrokes and because of the influence of those who want to replace assembly languages with high-level languages. We do not aim to promote CALM as a software development language. Rather, we believe clear notation also helps acquisition of assembly language expertise. It is profitable to understand the scope and limits of assembly language.

Since writing a Macintosh-like environment in CALM for the Smakys, we see that clear notation helps manage large programs. After a 60-hour course covering many other aspects, students develop very efficient notation skills. On the average, they can write a 12K-character source program and learn to document it in a very readable form. By using typographical commands and the various available tools, they produce nice printouts that even include screen copies meant to document the result inside the program.

Dauphin

Developed in 1976, the Dauphin, a didactic microprocessor system, used the low-cost Signetics 2650 microprocessor. Originally, members of a local electronics club obtained the Dauphin as a kit. Its modularity and use of the Mubus, a simple microprocessor bus, was a big advantage. The Dauphin system has been used to host more than 15 processors. Once a new processor board is inserted into a Dauphin, one can write and test the software, which is in some senses like testing the processor.

The advantage of the Dauphin—compared to the small 1976 Kim machine and the more recent Micro Professor system—is that its front panel shows bus activity and enables hand-controlled direct memory access and I/O. Figure 3 gives a better idea of the system. The bus includes 14 address lines and we now use 68008 and 68020 processors. The serial interface can download assembler and Modula-2 programs.

In my course, students just use the Dauphin to become familiar with processor operations. They access memory through both the front panel and the small debug monitor and go through the program step by step, exploring stacks and prefetch cycles.

The Dauphin keyboard allows the students to understand their first interface (Figure 4 on the next page). The keyboard only has 10 keys and reads as an 8-bit word. Students can easily check the priority encoder, and the full handshake flip-flop is especially easy to perform with the front panel. No latch is needed for the display, which must be refreshed. With the front panel, students can easily check the effects of data bits on I/O addresses 0 to 7 and understand the complete device register map.

Students use the hexadecimal monitor to introduce their first programs by hand. The loudspeaker generates sounds. Three nested loops produce a siren and enable access to two tables for playing music. Students quickly switch to downloading pre-prepared programs to learn more complex instructions before studying them. Then they use the Dauphin again in the middle of the course to test their first logidule interface (see box on page 65) with the front panel.

Hardware design course

The objective of this course is to acquaint students with the transistor-transistor logic (TTL) family, memories, and phase alternation lines (PALs). We require a one-semester

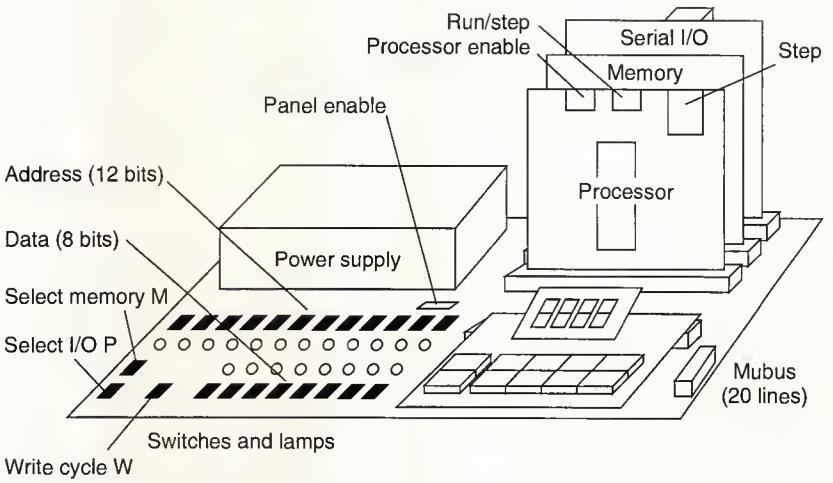


Figure 3. Diagram of the Dauphin system used by students.

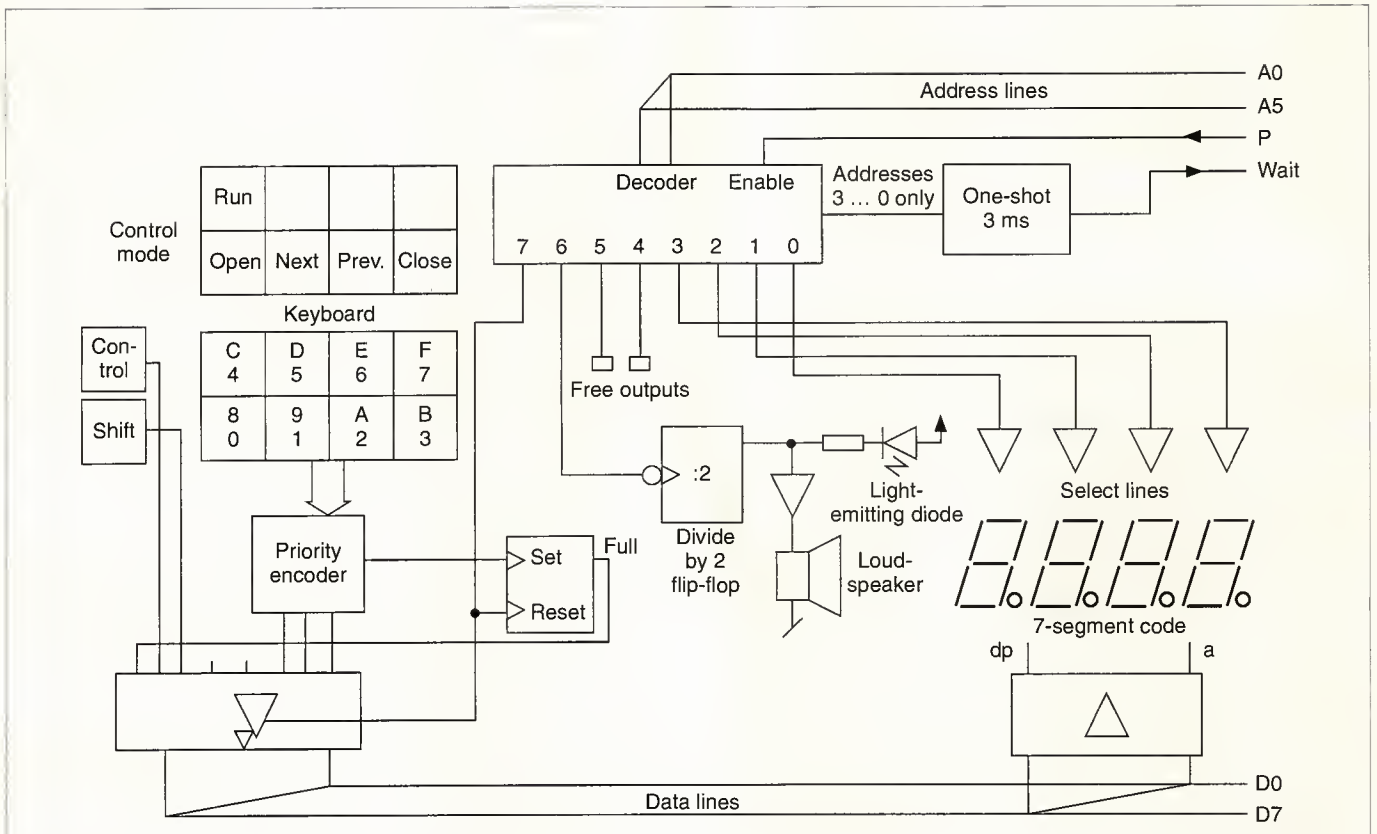


Figure 4. A Dauphin keyboard schematic.

logic system course before enrollment in this hardware design class.

Achieving the course objective is not easy in 40 hours. Consequently, we place instructional emphasis on practical laboratory work.[13] These labs, which last between one and four hours (although there never is enough time!), introduce:

- flip-flop and counter revision of basic logic carried out through questions and simulations on screen (1 hour);
- reflex measurement, period meter, and frequency meter (2 hours);
- registers and serial/parallel transfers (2 hours);
- latches and addressable latches, and output interface (2 hours);
- modulation, demodulation, and start/stop transfers (2 hours);
- graphic and alphanumeric, low-resolution video interfaces (4 hours);
- a nonvolatile serial memory interface (3 hours);
- a dynamic memory interface (3 hours);
- a PAL used to control a stepping motor (2 hours); and
- quadrature encoder and mouse interface (3 hours).

Some of these labs purely feature hardware. Others emphasize the hardware-software trade-offs (such as serial transfers), while still others focus mostly on software with an emphasis on real-time constraints (such as nonvolatile memory and the mouse interface).

A book now documents the theoretical portion of this course.[14-15] It serves as a reference document for any hardware designer, with clear and consistent symbols, synthetic surveys of basic features in ICs, and test problems.

A classified catalog within the book helps to select circuits. Redrawn logic symbols more clearly resemble our CALM approach. The ability to easily read the logic schematic language is important, especially since the industry does not always draw symbols in the clearest way. Figure 5 (on page 66) provides a few examples of these symbols. The book also thoroughly explains IEEE symbols, although we do not use them in the course.

We wanted the catalog to help the designer preselect components and establish detailed block diagrams and programmable logic device (PLD) design. With more than 1,000 standard ICs available, beginners confront a difficult task.

For the final schematic, the manufacturer's catalog has to provide the specifications of the selected ICs. The CAD tool used for entering the schematic and accomplishing

SUBSCRIPTION CARD READER-SERVICE CARD

IEEE MICRO

February 1991 issue (card void after August 1991)

Name _____

Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Country _____ Phone(_____) _____

Reader Interest
(Add comments on the back)

Readers,
Indicate your interest in articles and departments **by circling the appropriate number** (shown on the last page of articles and departments):

150	151	152	180	181	182
153	154	155	183	184	185
156	157	158	186	187	188
159	160	161	189	190	191
162	163	164	192	193	194
165	166	167	195	196	197
168	169	170	198	199	200
171	172	173	201	202	203
174	175	176	204	205	206
177	178	179	207	208	209

Product Information 1
(Circle the numbers to receive product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	
11	31	51	71	91	111	131	
12	32	52	72	92	112	132	
13	33	53	73	93	113	133	
14	34	54	74	94	114	134	
15	35	55	75	95	115	135	
16	36	56	76	96	116	136	
17	37	57	77	97	117	137	
18	38	58	78	98	118	138	
19	39	59	79	99	119	139	
20	40	60	80	100	120	140	

IEEE MICRO

February 1991 issue (card void after August 1991)

Name _____

Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Country _____ Phone(_____) _____

Reader Interest
(Add comments on the back)

Readers,
Indicate your interest in articles and departments **by circling the appropriate number** (shown on the last page of articles and departments):

150	151	152	180	181	182
153	154	155	183	184	185
156	157	158	186	187	188
159	160	161	189	190	191
162	163	164	192	193	194
165	166	167	195	196	197
168	169	170	198	199	200
171	172	173	201	202	203
174	175	176	204	205	206
177	178	179	207	208	209

Product Information 2
(Circle the numbers to receive product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	
11	31	51	71	91	111	131	
12	32	52	72	92	112	132	
13	33	53	73	93	113	133	
14	34	54	74	94	114	134	
15	35	55	75	95	115	135	
16	36	56	76	96	116	136	
17	37	57	77	97	117	137	
18	38	58	78	98	118	138	
19	39	59	79	99	119	139	
20	40	60	80	100	120	140	

SUBSCRIBE TO IEEE MICRO

All the facts about today's chips and systems

❑ **YES**, sign me up!

If you are a member of the Computer Society or any other IEEE society, pay the member rate of only $21 for a year's subscription (six issues).

Society: _____

IEEE membership no: _____

Society members: Subscriptions are annualized. For orders submitted March through August, pay half the full-year rate ($10.50) for three bimonthly issues.

❑ **YES**, sign me up!

If you are a member of ACM, ACS, BCS, IEE (UK), IEEE but not a member of an IEEE society), IECEJ, IPSJ, NSPE, SCS, or other professional society, pay the sister-society rate of only $38 for a year's subscription (six issues).

Organization: _____

Membership no: _____

Full Signature _____ Date

Name _____

Street _____

City _____

State/Country _____ ZIP/Postal Code

❑ Payment enclosed *DC residents: add sales tax*

❑ Charge to ❑ Visa ❑ MasterCard ❑ American Express

Charge-card number

_____

Expiration date

____ ____
Month *Year*

Prices valid through 12/31/91
2/91 MICRO

Charge orders also taken by phone:
(714) 821-8380 8 a.m. to 5 p.m. Pacific time
Circulation Dept.
10662 Los Vaqueros Circ., PO Box 3014
Los Alamitos, CA 90720-1264

Editorial comments

I liked: _____

_____

_____

I disliked: _____

_____

_____

I would like to see: _____

_____

_____

For reader-service inquiries, see other side.

PLACE POSTAGE HERE

PO Box is for reader-service cards only.

IEEE Micro
PO BOX 16508
NORTH HOLLYWOOD CA 91615-6508
USA

Editorial comments

I liked: _____

_____

_____

I disliked: _____

_____

_____

I would like to see: _____

_____

_____

For reader-service inquiries, see other side.

PLACE POSTAGE HERE

PO Box is for reader-service cards only.

IEEE Micro
PO BOX 16508
NORTH HOLLYWOOD CA 91615-6508
USA

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 38 LOS ALAMITOS, CA

POSTAGE WILL BE PAID BY ADDRESSEE

IEEE COMPUTER SOCIETY
PO BOX 3014
LOS ALAMITOS CA 90720-9804
USA

Logidules

In 1968, Hopt Electronic's Bipol, a German electronic construction game, consisted of 4 × 4 × 4-cm boxes with power supply contacts on the bottom and connection contacts on the top (see Figure D). Transistors and gates were proposed, but the functionality was very limited since connecting wires had to be made up using various kinds of boxes equipped with crossing and angled connections.

I began to use these boxes to demonstrate an iterative binary-decimal converter. With the help of my team and colleagues at the workshop of the Swiss Federal Institute of Technology (also called "Epfl"), I added new boxes that included most of the 74 IC family. The mold for these boxes is now in the hands of a Swiss partner. Each box holds four logical signals per side (eight for a double-size box), which is a perfect compromise. Most connections are established automatically; the 8-bit buses use flat cables.

The great advantage of logidules over Augat-like prototyping boards or over the racks found in many US universities is as follows:

- a logic symbol is visible,
- students handle essential wires only (power supply and buses connect automatically),
- the layout can grow at random in two directions, and
- contacts and wiring are reliable (due to the expensive connectors).

Of course, cost is a major problem, although the introduction of a new box design and the manufacture of larger quantities could significantly reduce the expense. The Epfl produces the majority of logidules. Students enrolled in the Interface or Microprocessor lab receive a box that includes 82 logidules of the following types:

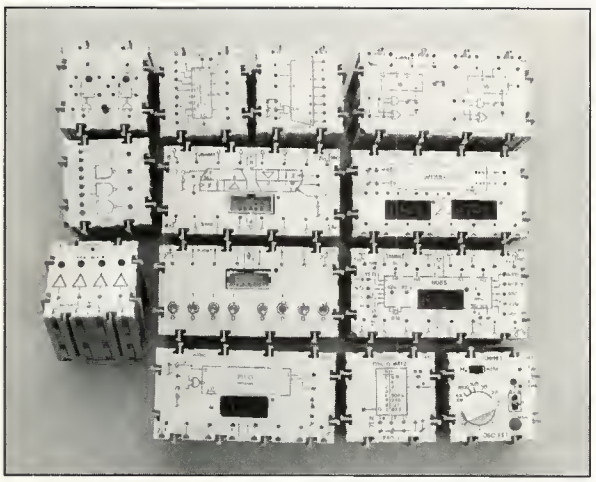


Figure D. Example of typical logidules.

- gates, flip-flops, and oscillators;
- stepping motor, DC motor with sensors, and keyboard;
- switches and lamps;
- counters;
- programmable interfaces;
- sockets;
- decoders, comparator, and ALU;
- shift registers;
- parallel registers and latches; and
- microprocessors and Mubus adapter.

We defined this set of logidules, taking into account lab needs. We included several logidules not normally used to represent a class of circuit (such as the priority encoder) and to familiarize students with what is available.

the layout will determine the way in which logic symbols will be finally documented. An additional layer of precise functional symbols is justified for both the novice and for the experienced designer.

Microprocessor interface design

When they start the microprocessor course, students are unfamiliar with component data sheets and technical English. They receive a document that explains both the key points in French and the original manufacturer's documentation in English. The lab enables students to really experience the way micros and programmable interfaces react.[25] Logidules and a Mubus 20 interface connected to the Smaky or the Dauphin emphasize any combination of hardware and software. Since each lab session lasts for only three hours, lack of time limits the experiments to the following:

- **8085 basic system** (see Figure 6 on the next page). Due to the logidule's automatic connection system, the students only wire one flat cable and about 10 wires to make it work. They become familiar with instruction steps, bus multiplexing, processor states,

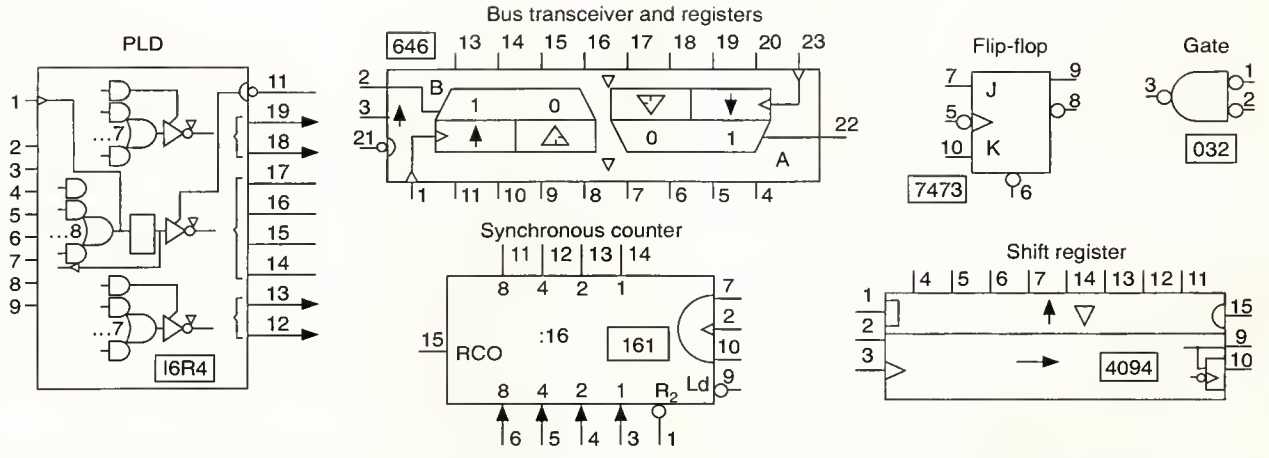


Figure 5. Sample of logic symbols.

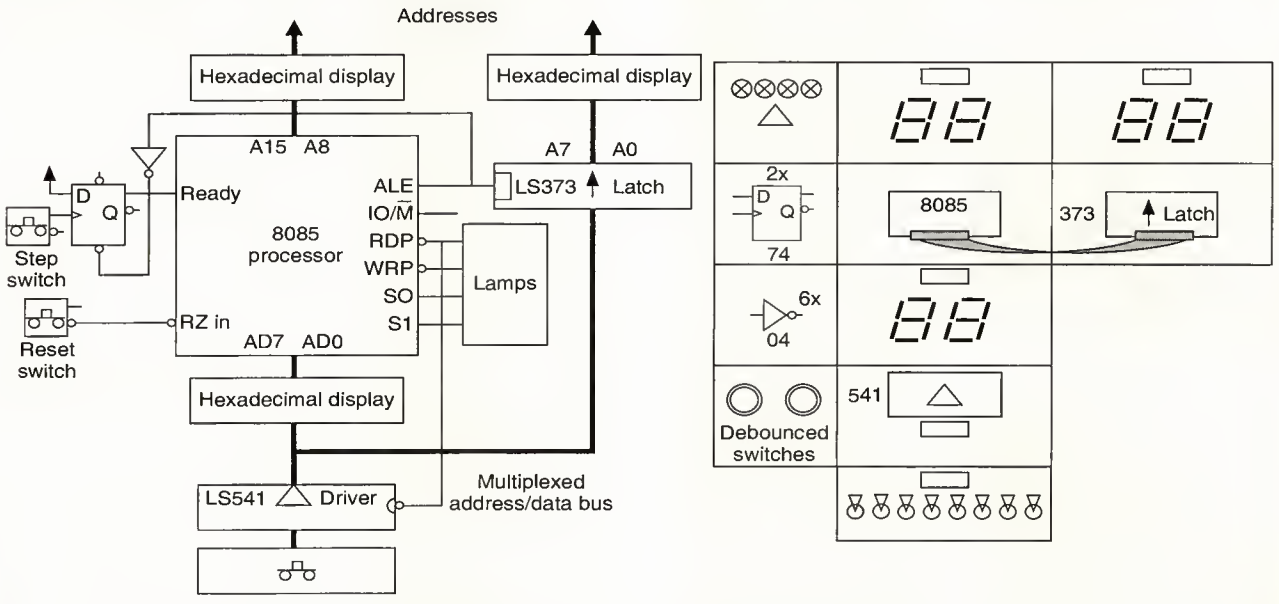


Figure 6. Diagram of an 8085 system with simulated instructions.

and timing diagrams. In the second part of the lab, they add a ROM and a RAM, and run test programs.

- **8085 interrupts.** Students discover the nature of a stack and learn to unmask interrupts and clear them.
- **8254 timer.** Lessons include checking the low-level internal synchronization mechanism, the access to registers, and the operation modes.
- **8255 parallel interface.** Students practice lighting lamps, reading a 4 × 4 keyboard and transferring data with handshaking between the two groups.
- **6801 microcomputer.** The session includes checking the existing test programs and creating a new piggyback EPROM (erasable, programmable read-only memory) with programs for the timer.
- **68030 screen routines.** Students learn to draw dots, lines and areas on screen and to check the difference between direct routines and system calls.
- **Modula-2 for I/Os.** Students learn to control a motor and read a 4 × 4 keyboard.
- **68030 system timing.** This laboratory session involves

the observation of memory and I/O signals on the Smaky 196, and the use of repetitive reset to see what happens with short-circuited lines, missing PALs, and enabled cache.

The laboratory meets once every two weeks, and runs in parallel with the course. It presents the most important features of some of the major processors (8085, Z80, 6801, 6809, 68020, and 8086) and programmable interfaces (6821, 8254, 8255, 8259, and 8536). It also covers RS-232 and SCSI microprocessor buses.

Advanced microprocessor course

We excluded practical work from this course for many years due to lack of time. For the first time this year, the Advanced Microprocessor Course allows students to choose one of three optional labs: DSP56001, floating-point unit 68882, or 68030 MMU. With additional time, we could organize labs on video RAMs, transputers, and 80386s.

Hardware projects

Our hardware projects are probably similar to those students work on elsewhere. We are lucky to have good local facilities for prototyping; each student can submit an interface schematic and receive the PC board back, usually within two weeks. At the end of the tenn, each student writes a report and a one-page summary, including a picture or block diagram. We distribute abstracts of these projects to colleagues and local companies. They are encouraged to submit similar project proposals to us for student involvement.

A selected list of recent microprocessor hardware projects, some of which last two consecutive terms, appear below:

- slow logic analyzer with a 6805 and liquid crystal display;
- control of five motors using a LM629 proportional-integrator-differentiator (PID) programable interface;
- network of 6805 processors with power interfaces for the control of a miniature railway network handling up to 600 switches and 80 engines;
- Dauphin processor card using the HD64180 processor;
- 68070 board as an SCSI/local network server;
- Ethernet interface using the DP83932;
- AT&T reconfigurable neural network interface; and
- S-bus to Mubus 20 interface.

THIS ARTICLE IS TOO SHORT to sufficiently detail our educational environment, and I did not mention all of the instructional accomplishments.

A software-oriented microprocessor and interface design course should offer students a large number of possibilities to experiment with various hardware and software environments. We built up a comprehensive set of experiments, most of which contain 40 copies. This set is adequate for dealing with 80 students at a time, supervised by six to eight assistants.

Designing and implementing computer equipment for educational purposes takes large amounts of money and many people. We are very grateful to our school for providing both of these resources. In a way, we are lucky to be in Switzerland: There is no local market, companies are conservative, and venture capital is difficult to find. This means we have plenty of time to develop new ideas and devote time to our students, instead of building companies.

References

1. R. Sommer, "Time-Sharing System of Computer-Like Peripherals for Teaching Minicomputer Use," *Proc. Conf. Computer Systems and Technology*, London, 1974, pp.106-110.
2. J.D. Nicoud, "A General Purpose Interface System," *Proc. Minicom Conf.*, Florence, Italy, 1973, pp. 8.1-8.14.
3. J.D. Nicoud and R. Sommer, "Modular Logic Elements, Microprocessors and Peripherals Improve Efficiency of Teaching and Development," *Proc. Compcon*, Spring 1975, *IEEE Computer Society* Press, Los Alamitos, Calif., pp. 127-130.
4. D. Dutoit, "Les microdules: ensemble de modules pour l'expérimentation avec des microprocesseurs et aide au développement" ["The Microdules: A Set of Modules for Experimenting with Microprocessors and Development Tools"], *Journées d'Electroniques*, EPF-Lausanne, Switzerland, Oct. 1974, pp. 207-214.
5. J.D. Nicoud, "Smaky, an Evolving Family of Personal Computers for Office Automation," *Second Workshop Office Information Systems*, North Holland, 1981, pp. 93-103.
6. H. Kirrmann, "The Smaky Story: The Swiss Personal Computer," *IEEE Micro*, Vol. 9, No. 4, Aug. 1989, pp. 78-79.
7. J.D. Nicoud, "Peripheral Interface Standards for Microprocessors," *Proc. IEEE*, Vol. 64, No. 6, pp. 896-899.
8. J.D. Nicoud and D. Del Corso, "Mubus Standard," *Micro Scope*, Lausanne, Vol. 1, No. 8, Apr. 1987.
9. J.D. Nicoud, "Standardized Mnemonics and Software Support for Microprocessors," *IEEE Int'l Symp. Circuits and Systems*, 1975, pp. 470-473.
10. J.D. Nicoud et al., "Software Facilities for Microprocessor Development and Teaching," *Proc. Euromicro*, North Holland, 1979, pp. 347-350.
11. J.D. Nicoud, *Microinformatique: architecture, interface et logiciel [Microcomputers: Architecture, Interfaces, and Software]*, Dunod, Paris, 1983.
12. J.D. Nicoud and A. Schmitz, *Introduction au 68000 avec SMILE [Introduction to 68000 using Smile]*, LAMI-EPFL, Lausanne, 1987.
13. J.D. Nicoud, *Laboratoires interfaces*, LAMI-EPFL, 1990.
14. J.D. Nicoud, *Circuits numériques pour interfaces microprocesseurs* Masson, Paris, to be published in 1991.
15. J.D. Nicoud, *Digital Circuits and Concepts for Microprocessor*

Interface Design, to be published in 1991.
16. R.D. Hersch, "Integrated Theory and Practice in a Microprocessor Systems Design Course," *Proc. Euromicro,* North Holland, Amsterdam, 1984, pp. 227-232.
17. J.D. Nicoud, "Microcomputer Buses and Links," *Encyclopedia of Physical Science and Technology,* Academic Press, London, 1990, pp. 440-449.
18. D. Del Corso, H. Kirrmann, and J.D. Nicoud, *Microcomputer Buses and Links,* Academic Press, 1986.
19. J.D. Nicoud and A.M. Tyrrell, "The Transputer T414 Instruction Set," *IEEE Micro,* Vol. 9, No. 3, June 1989, pp. 60-75.
20. J.D. Nicoud, "Video RAMs: Structure and Applications," *IEEE Micro,* Vol. 8, No. 1, Feb. 1988, pp. 8-27.
21. F. Wagner and J.D. Nicoud, "On the Notation of CALM—Common Assembly Language for Microprocessors," *North Holland Computer Standards and Interfaces,* North Holland, 1987, pp. 455-462.
22. J.D. Nicoud and F. Wagner, "Major Microprocessors: a Unified Approach Using CALM," *North Holland Computer Standards and Interfaces,* North Holland, 1987.
23. J.D. Nicoud and P. Fah, "Common Assembly Language for Microprocessors, CALM," tech. report, LAMI-EPFL, 1986.
24. J.D. Nicoud and P. Fah, "Explanations and Comments Related to the CALM Standard," tech. report, LAMI-EPFL, Lausanne, 1986.
25. J.D. Nicoud, *Laboratoires microprocesseurs [Microprocessor laboratories],* 3rd ed., LAMI-EPFL, 1987.

Jean-Daniel Nicoud is a professor at the Swiss Federal Institute of Technology of Lausanne, Switzerland. Active in microprocessor-related research for 23 years, he has designed many microprocessor-based systems. His interests, and those of his research group, include microcomputers, peripherals, graphics workstations, very large-scale integration, and neural networks.

Nicoud is a member of the IEEE Computer Society and was an associate editor of *IEEE Micro* from 1981 to 1985.

Address questions concerning this article to Jean-Daniel Nicoud, Swiss Federal Institute of Technology, Laboratoire de Microinformatique, EPFL-Ecublens, CH-1015, Lausanne, Switzerland.

Year 2000

continued from p. 13

Japanese researchers continued work in the field after IBM withdrew in 1983. AT&T continued a modest effort. Now both the National Commission on Superconductivity and a study sponsored by the US Defense Dept. Advanced Research Projects Agency have recommended that the US make a greater effort.

A group at the Hitachi Central Research Laboratory believes that "all of the techniques essential for a large-scale computer will be developed within several years." For the superconducting computer to become a practical product will require breakthroughs in circuitry, powering, and packaging. Thus, the Hitachi group concluded: "The Josephson computer might appear in the 21st century."[14]

Optical computing. Photons and electrons are little "somethings" that move at high speed and carry information. Electrons currently serve as the basic element in digital computers. Many scientists believe that photons can be harnessed for the same purpose. In fact, photons have been embodied in synthetic-aperture radar and other forms of optical signal processing. These kinds of processing, however, are not digital.

In another application, photons convey digital information over fiber optic links. The switches between these links are presently digital electronic, involving frequent conversions between photons and electrons. Establishing some optical switches at the linkage points would be more efficient.

Perhaps that is one reason why AT&T Bell Labs is one of the most active players in optical computing research. In January 1990 its research group, led by Alan Huang, demonstrated a prototype digital optical computer. Lasers generated the photons and optical on-off switches controlled them.

IBM, for one, has its doubts. "In the early and middle 1960s, researchers at IBM and elsewhere explored the potential of digital computing with photons instead of electrons," Trudy Bell reported.[15] "However, the digital techniques seemed impractical because of limitations in the physics and technology of materials and the devices that apparently could be built from them."

Of course, there have been significant developments since the 1960s, but the IBM researchers are still skeptical. "They feel that the problems encountered in the 1960s are basic to the physics involved, not merely the technology," Bell said.

A recent survey by the *New York Times* concluded that "scientists believe that optical on-and-off switches and even optical switchboards will one day replace their electronic counterparts in computers, although there is general agreement that practical [digital] optical computers are still many years away."[16]

Molecular computing. We have the existence proof in the form of organic life that protein molecules perform vari-

ous processing operations analogous to computing. For example, when foreign organic material enters the body, certain molecules recognize it as foreign and initiate a process that rejects it. Presumably a string of appropriately designed molecules in some artificial array could sense some phenomenon, "compute" what it means, and start other molecules working to do something about it.

Considerable biological research is underway all around the world and some research directed at this area commenced in 1983, according to Michael Conrad.[17] "It is likely that in the next several years some molecular devices will be exhibited that perform primitive information processing," he said. Biosensors are likely to be the first molecular application.

Of course, we need to learn much more about how life forms function, how to abstract out some very simple process from the complexity of evolutionary life, and how to construct the protein molecules to implement this process. "Marketable products seem decades away," Conrad concluded.

Mass storage

The race between mass-storage technologies continues. A decade ago observers thought that optical storage would take over some part of mass-storage applications because of its high bit density. On a different tack, as knowledge of Moore's law on density spread, chip enthusiasts looked forward to a time when some part of the data traditionally held in mass storage would migrate to main memory. As it happened, however, during the 1980s magnetic density kept pace with semiconductor density and gained on optical density.

Capacity of magnetic recording quadruples every three years (the same rate at which the capacity of dynamic random-access memory is increasing).—Roger Wood[17]

Magnetic storage density has been increasing by a factor of four every five years, a trend expected to continue.—NRC expert panel

Magnetic recording. The parameters that determine recording capacity include linear bits per mm, areal bits per sq mm, tracks per (radial) mm, data rate in Mbits per second, magnetic properties of the magnetic medium and the recording or reading head, signal processing techniques, and so on.

Let us look at one of Wood's[17] parameters—areal density. IBM's 1989 high-capacity 3390 magnetic rigid disk records at 96,000 bits per sq mm. The IBM Magnetic Recording Institute (San Jose, Calif.) demonstrated a density of 1.8 million bits per sq mm. "Some design challenges remain," Wood noted. "IBM has yet to announce commercial plans for the technology in a disk drive."

In spite of a history dating back to 1900, however, Wood sees no danger of stagnation in areal density advancements. "There do not appear to be any physical limits to prevent stretching densities to many megabits per square millimeter," he said.

Optical storage. Erasable optical systems currently record in the range of 400,000 to 700,000 bits per sq mm. Read-only compact disks record near 1 million bits per sq mm. The drawback to wider use of optical disks is not storage density. The difficulties in many products are relatively high cost, slow access time, and lack of erasability (or write capability). On the plus side optical disks are immune to head crashes and the disk is removable. Hence a great volume of data is separately stored or carried between computers.

Optical storage density is expected to increase at a rate slightly slower than that for magnetic technology.—NRC expert panel

Optical mass storage is a prime example of a technology with great promise that unfortunately began life in the wake of a continually developing technology (magnetic). Always a step or two ahead of the trailing technology, magnetic recording generated the necessary funds from expanding sales to continue its exponential rate of improvement.

Semiconductor storage. The areal density of a 4-Mbit dynamic RAM is 1.2 million bits per sq mm. Of course, future generations will have still greater density. Putting eight or nine of these 4-Mbit DRAMs on a card provides storage capacity equivalent to a 4-Mbyte hard disk, but with the direct-access advantage of semiconductor architecture. These semiconductor "disks" should become more competitive with hard disks during the 1990s as DRAM density increases and prices decline.

(Figure 2 shows Intel's new 2-Mbit Flash Memory devices, which illustrate the principle that when internal feature size

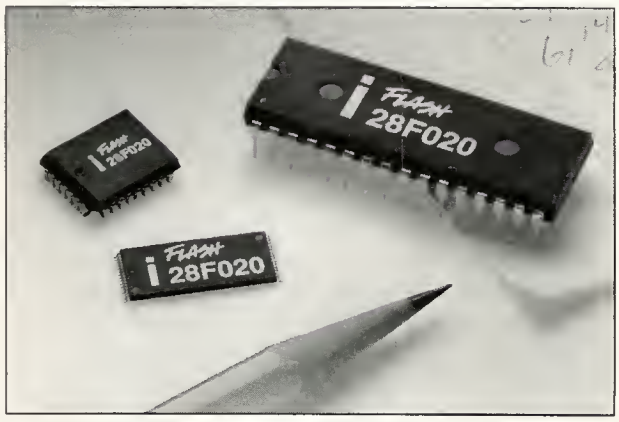


Figure 2. Intel's 2-Mbit Flash Memory devices.

drops, external dimensions decline, too. At 20 mm x 8 mm x 1.2 mm, a flash memory device can be mounted on each side of a memory card and still fit within the 3.3-mm thickness permitted by the governing standard.)

Of course, all three of these mass-storage technologies—magnetic, optical, and semiconductor—are "moving targets." As we contemplate an increase in the storage capacity based on one technology, we need to keep in mind that the other two technologies are also moving forward. In each case, a host of factors in addition to areal density—cost, reliability, perfomance, form factor, no moving parts, power consumption, and so on—will play a role.

The people interface

At present, the input to a microcomputer is overwhelmingly via keyboard; the output is a CRT display about half the size of a sheet of paper. The reason, of course, is that these two pathways to people are the least expensive. People have other ways of putting out information besides a keyboard—handwriting, diagrams, and voice. They also receive information in other ways, particularly speech and other sound signals. Moreover, their visual field is much larger than the small screen. So they might like to view a much larger screen.

Now all of these modalities are already in use, usually in a small way. More widespread use has been delayed by the need for additional hardware to accommodate them. That would raise prices. In addition, the algorithms to implement some of these interfaces—particularly handwriting recognition, voice recognition, and speech synthesis—are not yet entirely worked out. Moreover, it is not clear if people feel an acute need for these additional paths. At least, they have not surged eagerly to acquire the interface capabilities already on the market. Perhaps they believe the interfaces are too expensive for what they do.

The exponential growth of hardware capability, founded on the semiconductor laws, promises to provide the computing capability needed to implement these interfaces during this decade—at a cost much of the market will find bearable. It appears that much greater computing and memory capacity will be necessary to accommodate algorithms that adequate handwriting recognition, voice recognition, and speech synthesis will require. The neural-network principle is expected to do well at these tasks. The marketplace will decide whether people really want these added capabilities.

Handwriting recognition. Hand-printed capital letters (or other carefully constructed characters or symbols, perhaps in prescribed boxes on a form displayed on an entry tablet) can be recognized now. Several commercial products are on the market, with several more expected in the next year or two, including Windows H from Microsoft. Recognizing unconstrained cursive writing, Japanese and Chinese characters, and handwritten mathematical equations is much more difficult.

Professional writers who think that current word-processing technology puts too much machinery between their muse and their manifest words might adapt better to cursive-writing input. Just how they create the words seems to be a delicate subject for some creative writers. In particular, they might welcome the opportunity to edit a manuscript with a stylus, crossing out one word and writing in another in the traditional way. They would like to view more text than the small screen presently provides.

Similarly, mathematicians might seize upon a notebook computer that permits them to write a variety of symbols, superscripts, and subscripts in their usual manner.

Moreover, many professionals like to make a point with a diagram. Certainly one could use a stylus to draw a diagram and store it in pixel form. How it could be further processed and for what purpose remain to be invented.

The drawing, storing, and processing of kanji characters is an analogous problem. Naturally, this greatly interests the Far East.

All these difficult tasks are mainly a matter of developing complex algorithms. Consequently, predicting the progress made by 2000 for such efforts is more chancy than to forecast increased transistor density. At least the hardware capability to implement the new algorithms inexpensively will become available during the decade.

By 2000, much more powerful hardware will accommodate more sophisticated algorithms at a lower price.

Voice recognition. Inputting information to a computer by voice would be even easier than by handwriting. The capability has been available, at least in rudimentary form, since the mid-1970s. I remember reporting on such a system at that time. Aircraft maintenance was one of the applications. The mechanic, both hands at work, audibly reported part numbers and conditions to a central computer. The system accepted only a limited vocabulary, required each spoken word to be separated, and had to be trained to each user.

This system was a far cry from a secretary to whom you could casually direct a comment in colloquial speech. Performance has improved since then, but "comfortable and natural communication in a general setting (no constraints on what you can say and how you say it) is beyond us for now, posing a problem too difficult to solve," said Richard D. Peacocke and Daryl H. Graf of Bell-Northern Research.[19] "It seems likely that we will also need natural language understanding before we

can achieve comfortable and natural communication with computers through voice."

So, again, we have to know more about the patterns of speech on the semantic level. It will take complex algorithms to decipher them, which will likely require a lot of computer power. In the meantime, relatively simple applications will continue to grow.

Speech synthesis. Text-to-speech conversion is now in common use in reading devices for the blind and in voice mail systems. However, it has not taken the personal-computer world by storm. Very likely sighted persons would just as soon, under routine circumstances, read messages as hear them.

"Our current understanding does permit the synthesis of intelligible speech," Michael H. O'Malley[20] wrote recently, "but our models, especially in their dynamic behavior, are not yet adequate to make synthetic speech that is indistinguishable from human speech."

"Generally speaking, text-to-speech systems are limited by our current knowledge of linguistics," he continued. "The resulting speech sounds somewhat boring, but most attempts to make it more lively result in some sentences having a foolish-sounding prosody."

As to the future, O'Malley said that "unrealistic expectations for dramatic future improvement ... sometimes arise from an unsophisticated view of the complex linguistic information involved. Improvement will continue at the same slow, steady pace that has produced incremental progress in accuracy, intelligibility, and naturalness over the past three decades. However, we can expect faster progress in methods of delivering the technology."

By 2000, much more powerful hardware will accommodate more sophisticated algorithms at a lower price. The growth of speech synthesis is likely to revolve around finding applications that people find useful.

Displays. What is the wish list in this area? It includes color, brightness, higher resolution, double-page screens with windows capabilities, capability to move images and three-dimensional drawings, viewing angle, response time, light weight, low power, ruggedness, portability, and minimization of radiation. Most of these attributes are already available—at a price. During the next 10 years, because of advances in hardware, the price per attribute is certain to decline.

CRTs will continue as the display of choice over the next few years, according to a recent report by the Office of Technology Assessment (OTA).[21] In the longer term, however, the Office expects a shift away from CRT to technologies such as liquid crystal, electroluminescent, or plasma displays. Less well-known display technologies are also under investigation.

"By the late 1990s, yet another display technology may become available—the active matrix flat panel liquid crystal display," OTA speculated. Ten- to 14-inch color displays of 640 × 400 pixels have been demonstrated, mostly by Japanese companies.

Display size (diagonal) is increasing at the rate of three inches (75 mm) per year.—OTA

Commercial production is about three years behind research and development demonstrations. High-definition television, when it takes off, will create the market to support a large-screen, high-resolution display. It will then be available for computer applications, too.

Making it happen

Outlining where the semiconductor laws and other advances might take us by 2000 was the easy part. Now comes the hard part. We, the human race—or at least some part of it—need to make it happen.

Maintaining the pace of technology development during the next 10 years will take huge investments. While the immediate funds may seem to come from investors, banks, and company profits, they come from the marketplace, directly or indirectly in the ultimate analysis. Products have to be sold to large markets and the proceeds reinvested in development.

A growing market for computer products requires new applications, additional users, or more intensive use by existing users (Figure 3). Where are the mass-usage applications comparable to word processing or spreadsheets? Where are the new users? How can we encourage present users to do more?

The answers to these questions generally involve software. Some answers may lie in integrated systems, easy-to-use interfaces, open systems, and standards. If appropriate com-

Figure 3. Weighing less than 7 pounds, the Zeos Notebook Computer represents the kind of personal computer technology available at the beginning of this decade.

munities of manufacturers, marketers, and users can agree on these matters, software developers can produce the implementing software. But developers cannot program consistent people-computer interfaces, for example, if they cannot agree on what they are.

Now, all of this involves people seeing the possible ways to accommodate the semiconductor evolution, agreeing what these ways should be, and then learning to use them. That takes funds. Moreover, it takes time—usually time measured in years. A relatively small engineering group may develop the next-generation microprocessor quite rapidly. Getting people to adopt the resulting paradigms is a slower process. Yet it takes funding from a large number of people to support the development groups and new fabrication facilities.

Huge investment. The money necessary to keep the semiconductor industry on its exponential growth curve has also been increasing on its own exponential path. Twenty-three years ago Intel developed two new technologies—silicon-gate MOS and Schottky bipolar—for about $2 million, Moore remembered. Today, the cost of developing a new technology is around $60 million. Intel recently announced that its capital budget for 1991 is $600 million. The cost of building a fabrication facility is in the $200- to $400-million range. Later in the decade, because of the increasing complexity of denser processes, that facility cost may climb to $500 million. An X-ray process in the late 1990s may reach one billion dollars.

Even large firms are finding it necessary to join forces to develop next-generation technologies. The other side of these costs is that recouping the investment requires heavy product sales. To put it another way, companies must justify the funds necessary to finance the next generation.

> **The number of transistors produced doubles every year.—Moore**

This is approximately what all these exponential curves mean in terms of market expansion. The production curve is currently going through about four million transistors per person in the industrialized world.

> **Manufacturing cost per chip will begin to increase as feature size penetrates the fractional micron range, but cost per unit capability will continue to decline.—NRC expert panel**

What is the industry going to do to sell transistors in these incredible quantities and at higher chip prices? Well, one outlet is new applications.

New applications. Just three application programs account for the bulk of personal computer software: word processing, spreadsheet, and database management. Tens of thousands of application programs divide the rest of the sales among themselves. An estimated 40,000 programs run on the 8086. Perhaps desktop publishing is the most recent big application area to arrive on the scene. Another is laser printing; it takes a lot of transistors to print in many fonts.

"If it doesn't do something that the people are doing right now in great frequency, it couldn't possibly be that important," Charles Simonyi of Microsoft Corp. declared in a September 1990 theme issue of *Byte* largely devoted to the next 15 years. Well, one could amend that statement to say "what people *want* to do right now," but the computer power and programs are not yet available.

> **We've seen the price per instruction per second, which dropped at a rate of roughly 15 percent a year for the previous two decades, drop by more than 50 percent a year over the last several years.—Stephen C. Johnson[22]**

One activity in which people do engage is television watching. Combining television with a personal computer may result in an application area that absorbs some of those millions of transistors. This combination already has a name: multimedia.

Multimedia has been available from Intel, for instance, in the form of add-on boards. Originally the seven add-on boards cost about $22,000. In February 1990, Intel reduced the system—which it calls digital video interactive (DVI)—to two boards for $2,000. Then last November Intel released a pair of microprocessor chips (750) with DVI for about $100. DVI enables personal computers to present full-motion video, high-resolution still photographs, animation, and stereophonic sound. More than 200 companies are developing products based on this set. The price of a multimedia-equipped personal computer—expected to cost an additional $1,000—will soon be within the range of many consumers.

Storing multimedia digital data will require either enormous quantities of mass storage or high-speed data compression and reconstitution. The latter solution will use more millions of transistors.

Another promising application goes by the name of groupware. As defined by Terry Winograd[23] of Stanford University, groupware is "a new kind of software that supports cooperative work." Other participants in this new field call it coordination theory, collaboration technology, or computer-supported cooperative work. For the most part, activity in this field is still at the research stage—trying to figure out the mechanics of how people work together in groups. Still, a dozen or so ideas have been reduced to actual software and some are for sale commercially.

For example, an electronic meeting system developed at the University of Arizona with IBM support is intended to get more ideas out in the open. In hierarchical organizations, lower ranking participants are often loathe to express their opinions freely. The electronic meeting system permits them

to interject their ideas anonymously via keyboard.

Some of the best minds in the software business are trying to find sizable new applications. Thousands of application programs are jostling for position in today's marketplace, with only a few becoming commanding applications. No doubt one or two more sizable applications will turn up by 2000.

More intensive use. Since uncovering market-expanding software applications is tough, the next recourse is to enlarge the market by encouraging present users to use their computers for additional tasks. Niche programs provide reasons for more intensive use.

However, practical obstacles slow this expansion process. One obstacle is the lack of standardization between program interfaces. That is, everything from function-key actions to little procedures differs from one program to another. That makes it hard for people to use as many different programs as their work might justify.

Some of the best minds in the software business are trying to find sizable new applications.

Some developers have tried to make program interfaces more similar to each other, that is, to develop "integrated" systems of several programs. Various forces have hampered these attempts, including product differentiation, rapid release of new generations, and copyrighting of "look and feel."

Electronic mail is an example of an application that could be used much more extensively. Everyone who wants a substitute for telephone tag is a potential customer for it. Just about everyone has a telephone, but the estimated number of people who actually send messages from their home terminals at least once a week is only in the hundreds of thousands.

At the present, several factors are slowing the growth of e-mail. First, it is too hard to use. The complex procedures to address a recipient are even more complex if that recipient is on another network. There is no universal directory of e-mail addresses. And—a chicken-or-egg problem—relatively few of one's correspondents use e-mail. Ideally e-mail should work as easily as the telephone system. Parenthetically, it took the US telephone system 40 or 50 years to move from scores of independent systems to one integrated system.

Another obstacle is the conflict between proprietary systems and open systems. Some companies see an advantage in locking in their customers with proprietary software. From the standpoint of growing the overall market, however, open systems seem to be more effective.

Integrated systems rest on a structure of standards. Building that structure takes time.

> **The cost/performance or size/performance of algorithmically specialized functions such as signal processing, database management, and flow model processing will be improved by the year 2000 by factors of two to three orders of magnitude, primarily through the use of parallel and distributed architectures.—NRC expert panel**

Expand the market. The bulk of computer systems sales today takes place in the industrial world, mainly the United States, western Europe, and Japan. One billion consumers live in these countries, and not all of them have yet been reached. More than four billion people make up the developing world. In time they will absorb great quantities of transistors.

Time

People and organizations resist change. Some are constitutionally resistant. Others are willing to change, but the process itself takes time.

Therefore, the technology development outlined in this article may proceed more slowly than technically possible. Shortage of funds may slow progress. The shortage may result from an inability to increase the market enough to generate the necessary funds to develop technologies. The market growth may be slower than hoped for because great new applications do not get discovered. In any case most of the people and organizations may decide to advance at a slower pace than the technology could theoretically allow.

IN NATURE, EXPONENTIAL GROWTH CURVES always top out. They turn into S-shaped curves. The fundamental reason is that the expanding species outstrips the environmental resources needed to support the next increment of expansion. For the same reason the rapid, exponential expansion of the computer industry will slow down when it outstrips its resources, or whenever the market fails to supply exponentially increasing resources.

I know you hoped I was going to lay out the next 10 years concretely—so you could plan on it! Instead I have offered glimpses of a much booby-trapped obstacle course, littered with conundrums about what people and their organizations may do.

The technical challenges ahead of us in this decade are as great as they ever were in past decades. The "end of history"—a thought that was briefly popular a year or two ago—is not yet. The technological future is almost boundless, the

"laws" tell us, certainly for this decade. Many organizations that understand these possibilities and are willing to finance further investment, seek broad-ranging applications, open new markets, and exploit present markets more intensively will grow and prosper.

References

1. S.F. Lundstrom and R.L. Larsen, "Computer and Information Technology in the Year 2000—a Projection," *Computer*, Vol. 18, No. 9, Sept. 1985, pp. 68-79.
2. F. Masuoka, "Are You Ready for Next-Generation Dynamic RAM Chips?" *IEEE Spectrum*, Vol. 27, No. 11, Nov. 1990, pp. 110-112.
3. P.P. Gelsinger et al., "Microprocessors Circa 2000," *IEEE Spectrum*, Vol. 26, No. 10, Oct. 1989, pp. 43-47.
4. L.R. Morris, "Guest Editor's Introduction to Digital Signal Processing Microprocessors," *IEEE Micro*, Vol. 6, No. 6, Dec. 1986, pp. 6-8.
5. S.A. Dyer and L.R. Morris, "Guest Editors' Introduction to Floating-Point Digital Signal Processing Chips," *IEEE Micro*, Vol. 8, No. 6, Dec. 1988, pp. 10-12.
6. H. Davis, R. Fine, and D. Regimbal, "Merging Data Converters and DSPs for Mixed-Signal Processors," *IEEE Micro*, Vol. 10, No. 5, Oct. 1990, pp. 17-27.
7. E.A. Lee, "Programmable DSPs: A Brief Overview," *IEEE Micro*, Vol. 10, No. 5, Oct. 1990, pp. 14-16.
8. J.J. Hopfield, "Artificial Neural Networks," *IEEE Circuits and Devices*, Vol. 4, No. 5, Sept. 1988, pp. 3-10.
9. M. Walker, P. Hasler, and L. Akers, "A CMOS Neural Network for Pattern Association," *IEEE Micro*, Vol. 9, No. 5, Oct. 1989, pp. 68-74.
10. W. Myers, "Artificial Neural Networks Are Coming," *IEEE Expert*, Vol. 5, No. 2, Apr. 1990, pp. 3-6.
11. R. Cates, "Gallium Arsenide Finds a New Niche," *IEEE Spectrum*, Vol. 27, No. 11, Apr. 1990, pp. 25-28.
12. R. Cates, M.J. Helix, and K.V. Rosseau, review of "Gallium Arsenide Digital Integrated Circuit Design," *IEEE Spectrum*, Vol. 27, No. 11, Nov. 1990, pp. 11-12.
13. A. Pollack, "Making Computers a Million Times Faster," *New York Times*, Sept. 26, 1990, p. C7.
14. Y. Hitano et al., "A 4-Bit, 250-MIPS Processor Using Josephson Technology," *IEEE Micro*, Vol. 10, No. 2, Apr. 1990, pp. 40-55.
15. T.E. Bell, "Optical Computing: A Field in Flux," *IEEE Spectrum*, Vol. 23, No. 8, Aug. 1986, pp. 34-57.
16. M.W. Browne, "Magic Crystals: Key to New Technologies," *New York Times*, Feb. 20, 1990.
17. M. Conrad, "The Lure of Molecular Computing," *IEEE Spectrum*, Vol. 27, No. 5, Oct. 1986, pp. 55-60.
18. R. Wood, "Magnetic Megabits," *IEEE Spectrum*, Vol. 27, No. 5, May 1990, pp. 32-38.
19. R.D. Peacocke and D.H. Graf, "An Introduction to Speech and Speaker Recognition," *Computer*, Vol. 23, No. 8, Aug. 1990, pp. 26-33.
20. M.H. O'Malley, "Text-To-Speech Conversion Technology," *Computer*, Vol. 23, No. 8, Aug. 1990, pp. 17-23.
21. Office of Technology Assessment, *The Big Picture: HDTV and High-Resolution Systems*, Government Printing Office, Washington D.C., June 1990.
22. S.C. Johnson, "Hot Chips and Soggy Software," *IEEE Micro*, Vol. 10, No. 1, Feb. 1990, pp. 23-26.
23. T Winograd, "Groupware: The Next Wave or Just Another Advertising Slogan?," *Digest of Papers*, IEEE Computer Society Compcon, Los Alamitos, Calif., Spring 1989, pp. 198-200.

Ware Myers has been a writer on computer technology since 1976. In addition to serving as a contributing editor of *IEEE Micro* and other magazines of the Computer Society, he has performed assignments for Xerox, Quantitative Software Management, Microelectronics and Computer Technology Corporation (MCC), Electronic Data Systems, and other corporations.

For 15 years at Xerox Data Systems, Scientific Data Systems, and Consolidated Systems Corporation, Myers worked in engineering groups developing analog-to-digital converters, digital-to-analog converters, color display stations, a microprogrammed controller, metal-oxide semiconducting memories, and other digital products. He was a lecturer in engineering management at the University of California, Los Angeles (UCLA) and is the author of more than 150 articles.

Myers received a BS from Case Institute of Technology and an MS from the University of Southern California. He is a member of the IEEE Computer Society and Tau Beta Pi.

Address questions concerning this article to Ware Myers, 1271 North College Avenue, Claremont, CA 91711.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 150 Medium 151 High 152

Fun and games

continued from p. 21

and the need for calibration. They also learn about the data strobe and other handshaking built into the Centronics protocol.

***X-Y* drill positioner.** Figure 5 shows the interface signals that connect the 68KECB to an *x-y* drill positioner. Pulse outputs on PB0-3 on the MC6821 PIA control the *x-y* positioning of the drill bit over a fastened plastic board. Power outputs on PB4 and 5 control both the lowering of the drill mechanism and rotation of the drill bit. (In practice, for safety reasons, we fasten a pencil to the drill housing to trace a dot pattern on a piece of paper affixed to the upper side of the plastic board, rather than use a drill bit.) Feedback signals from the *x*- and *y*-movement of the drill mechanism accurately position the drill bit over the paper-covered plastic board.

In this experiment, students "drill" selected patterns, such as their name or initials, the Australian flag, the University of Wollongong logo, a star or a circle of specified diameter (and fixed spacing of holes around its perimeter), or a similar symbol. Feedback pulses arrive back from the drill at the rate of around 330 per second. Counting these pulses is necessary to position the drill bit accurately. An important real-world obstacle the students have to take into consideration here is "overshoot"—the reception of a few pulses after the motor has turned off. When overshoot is not factored in, the *x-y* position is lost.

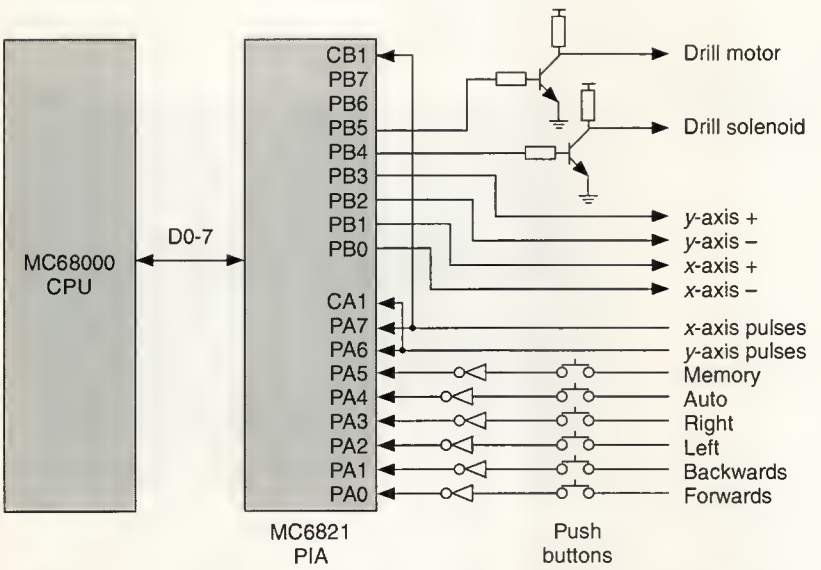


Figure 5. Diagram of an *x-y* drill positioner exercise.

We load the drill's data register with zeroes initially to ensure it always starts from a known position; the drill is positioned at its top left-hand corner, both before and after drilling. Furthermore, students need to allow for spurious interrupts caused by noise spikes generated by the solenoid.

Turtle robot. The mechanical construction of a turtle robot is shown in Figure 6a and b. Separate drive motors control its two wheels. Since no feedback is provided, stepper pulses must be counted to keep track of the turtle's position. Students can alter the turtle's speed by varying the duty cycle of the square wave of these stepper pulses. Students derive the stepper pulses/distance conversion factor empirically.

The four bumper sensors evenly spaced around its perimeter provide feedback. Students can use this information to detect collisions, back off, and then try an alternative approach. This is especially important in finding a path through a maze, and indeed can serve to introduce students to simple machine learning algorithms, as alluded to earlier. The appropriate initial action for responding to hitting an obstacle is as follows:

```
if_obstacle_hit
        stop_motors
        speak     /* using onboard Digitalker speech chip */
        print_warning_message_to_screen
```

Also fitted on the turtle are two "eyes" (light-emitting diodes), a loudspeaker connected to a Digitalker speech synthesis chip, and a solenoid that raises and lowers a pen.

The turtle interfaces to the controlling 68KECB via an R6551 Asynchronous Communications Interface Adapter (ACIA) use another experiment pod (serial I/O). In this exercise, we introduce students to the fundamentals of command-line interpreters, since they must effectively reproduce their own Logo-like language interpreter for the turtle.

A typical movement of the turtle would be as follows:

```
(> LEARN)
>E (turn eyes on)
>P (lower pen)
(The following seven movements trace
out a square of a side measuring 20
cm.)
>F 20 (forward 20 cm)
>R 90 (turn right through 90°)
>F 20 (= both motors clockwise)
>R 90 (l_motor clockwise; r_motor
counterclockwise)
```

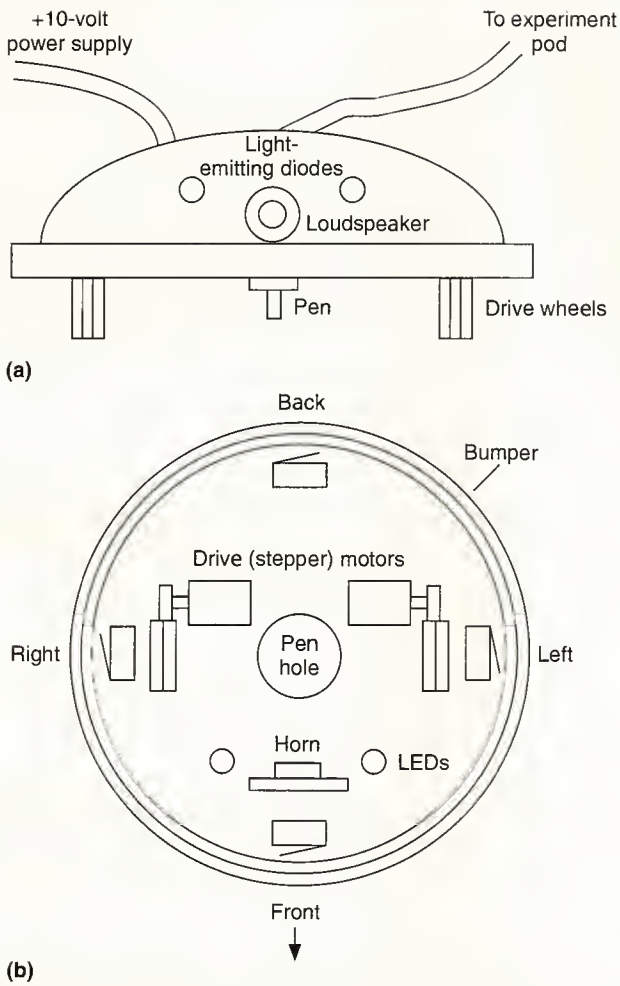


Figure 6. Front external view (a) and internal design (b) of a turtle robot.

```
>F 20
>R 90
>F 20
>P (toggle pen—raise)
>E (toggle eyes—turn off)
(> END (store above movements in memory for repetition
        at some later time)
```

Bit-mapped graphics. Figure 7 shows the laboratory setup used to interface a low-resolution, bit-mapped graphics device to the 68KECB host. The graphics output device is formed from a modified Volker-Craig VC404 terminal—modified in the sense that the character-generator ROM can be bypassed to access individual screen pixels. The graphics experiment pod contains an MC6845 CRT controller chip, together with 16-Kbyte words of video refresh RAM. Successive memory cycles alternate between the CPU and CRT chip. The light-pen input capability of the MC6845 is not used in this experiment; students must write their own graphics primitives:

- blank (fill) screen (a 6845 attribute);
- move_to (using pointers);
- draw_line (incorporating Breshenhem's algorithm);
- draw_rectangle (again, using pointers: TL, BR);
- draw_circle
- text (and so on...).

Then we use these primitives to create patterns on the VC404 screen, such as a Maltese or St. John's cross, the Olympic symbol, an Australian flag, or the University of Wollongong logo. Figure 7 illustrates how students become inspired to produce software that goes beyond what is originally asked of them. A student, inspired by the possibilities offered by the experimental hardware available for projects, produced the output on the CRT screen display shown in Figure 7. The main obstacle to overcome in this experiment is the mapping of the video refresh memory to the nonsequential pixel organization of the CRT screen.

Data logger. The data logger experiment pod uses an MC6821 PIA to interface the Heathkit ID4001 Digital Weather Computer to the 68KECB. The Weather Station computer can display indoor/outdoor temperature (Centigrade, Fahrenheit), time, date, wind direction, speed (mph, kph), and barometric pressure (inches/millibars). Thus, students learn about multiprocessing in this experiment, since the MC68000 and the MK3870 microcomputer must interact here.

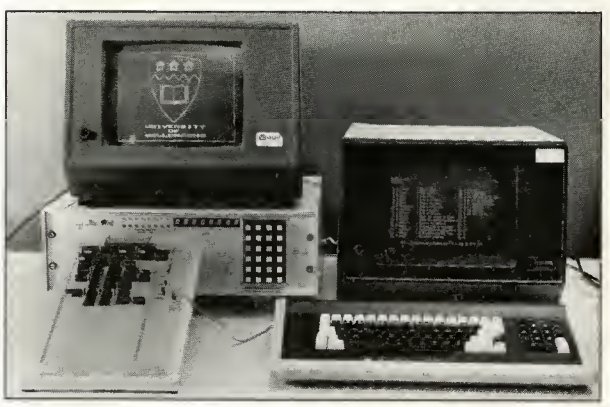


Figure 7. Example of low-resolution graphics capability using the University of Wollongong logo and appropriate equipment.

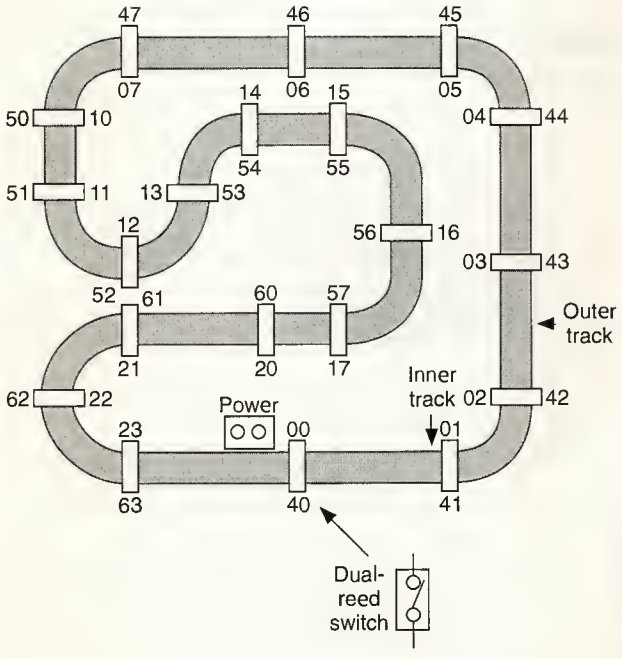


Figure 8. Example of a slot-car track design with mercury-reed switches. Note: Numbers reflect position on the switch matrix.

Interfacing is complicated by the fact that the Weather Station outputs strobed, 7-segment display data, which must be decoded within the control software. We ask students to first dynamically display the various Weather Station outputs on the terminal screen (using 68KECB firmware routines). Second, students log readings independent of the ID4001 for later statistical analysis and display. These readings include minimum, maximum, and average values; and times of day at which they occurred—a feature not found on the Weather Station.

Slot-car controller. We developed a slot-car set interface that uses a D/A converter to control the car's speed and discrete latches to record the car's position on the track. Closure of mercury-reed switches placed at regular intervals around the track indicate the car's current position. Figure 8 shows the positioning and numbering of these reed switches; they connect in a row/column matrix.

Students consult a lookup table to determine the appropriate car speed for the upcoming section of track. Students can either load these speed values during program initialization (in other words, static), or be dynamically updated based on past experience (by incrementally increasing the speed until the car crashes!).

Students can actually write their own primitive artificial intelligence software by incorporating a learning algorithm for the appropriate speeds for the various sections of the track:

```
set_all_speeds_to_default
/* i.e., start slow, & increase speed in discrete steps */
repeat
        for track_section = 1 to 20 do
                increase speed by (fixed) factor
                if next_reed_switch not crossed
/* i.e., too fast for that section (car crashed) */
                decrease speed by (fixed) factor
until reset
```

In this exercise, students wrestle with another obstacle—determining what is the correct response to a faulty (nonoperational) reed switch during the car's lapping of the track.

THESE LABORATORY EXERCISES have proved not only popular with students over the years but are also quite instructive. Indeed they often help the student excel in subsequent projects, as part of the Honors/Master of Science courses. Typical projects include a controller for a 5-degree-of-freedom robot arm[8] and algorithms for model railway scheduling.[9]

We've found that if one can spark students' interest and imagination, they readily absorb and retain real-time computing principles. Moreover, the students enjoy themselves throughout the process!

References

1. J.A. Fulcher, *An Introduction to Microcomputer Systems: Architecture and Interfacing,* Addison-Wesley, Reading, Mass., 1989.
2. J.A. Fulcher, "On the 'Best' Way to Teach Microcomputer Interfacing," *Proc. Fifth IFIP World Conf. Computers In Education,* Australian Council for Computers in Education, Sydney, July 1990, pp. 100-101.
3. J.A. Fulcher, and M.J. Milway, "The Development of a Microcomputer Laboratory for the Teaching of Real-Time Computing," *J. of Electrical & Electronic Engineering, Australia,* Vol. 8, No.3, Sept. 1988, pp. 159-167.
4. J.A. Fulcher, "Laboratory Support for the Teaching of Computer Architecture," *Int'l. J. of Applied Eng. Education,* Vol. 5, No. 2, 1989, pp. 229-238.
5. *MC68000 Educational Computer Board User's Manual,* Motorola, Phoenix, Ariz., 1982.
6. R.W. Floyd, "The M68000 Educational Computer Board," *Byte,* Vol. 8, No.10, Oct. 1983, pp. 324-336.
7. P.J. McKerrow, "Microcomputers, Slotcars, and Education," *IEEE Micro,* Vol. 3, No. 1, Feb. 1983, pp. 62-65.
8. A. Zelinsky, "Robotics Software for the Minimover-5 Robot

Arm," Dept. Computer Science Preprint #84-6, University of Wollongong, Australia, 1984.
9. A. Cartwright, *Model Railway Scheduling Algorithms,* Diploma computing science thesis, University of Wollongong, 1988.

John A. Fulcher is a senior lecturer in computer science at the University of Wollongong. His teaching responsibilities include computer architecture, computer interfacing, real-time computing, computer systems, and neural networks. The author previously worked as an engineer for Telecom Australia and lectured in electrical engineering at both Swinburne Institute of Technology, Melbourne, and Ballarat College of Advanced Education, Victoria.

Fulcher holds an honors degree in electrical engineering from the University of Queensland and an MSc from La Trobe University in Melbourne. He is currently working towards a PhD in neural networks at the University of Sydney. He is the author of *Microcomputer Systems—Architecture and Systems,* and is a member of the IEEE Computer Society, ACM (and its special interest group on architecture), and International Society of Neural Networks.

Address questions concerning this article to the author at Department of Computer Science, University of Wollongong, PO Box 1144, Wollongong, NSW 2500, Australia; or e-mail at john@cs.uow.edu.oz.

AEMS

continued from p. 25

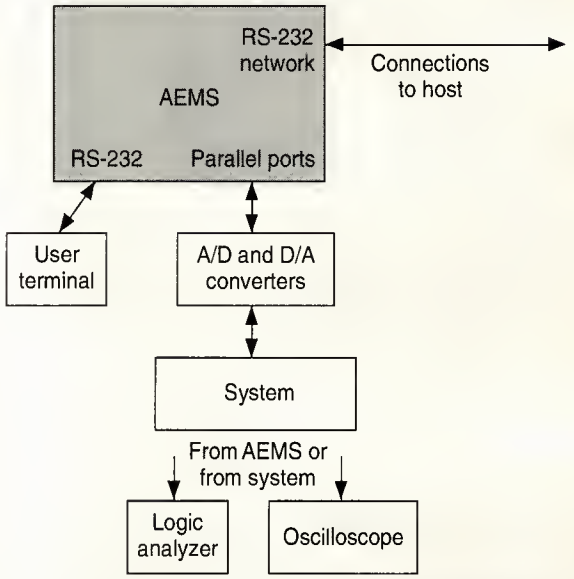


Figure 4. The AEMS setup for teaching digital signal processing and control system.

robotics.[11,12] Figure 4 presents a typical setup. The parallel ports shown in Figure 4 carry information to and from the AEMS and interface to an A/D and D/A system, but other interface mechanisms could be used. Once again, the students can use the oscilloscope and logic analyzer to view activities in the AEMS or in the system under evaluation.

This platform will be useful in the development, execution, and evaluation of real-time control and signal processing algorithms. For instance, students can use it to test and evaluate design algorithms for FIR (finite-length impulse response) and IIR (infinite impulse response) filters, as well as to evaluate the different filter realizations of transversal, lattice, and escalator filter structures. Because of the computing power available and the interfaces provided in the AEMS board, students can conduct challenging real-time experiments, such as adaptive signal processing implementations.

In applications such as the control of robots, students use digital controllers, though the system to be controlled is analog. Digital controllers are more versatile, cheaper, more accurate, and more reliable than their analog counterparts. The introduction of these digital controllers (filters) into a feedback configuration, however, creates a host of problems that students can investigate with the present board. For example,

the choice of a sampling interval and its effect on the control system are crucial to its closed-loop performance. With this board, students may experiment with many sampling intervals and display and analyze the affected signals. Another important issue in digital systems deals with the word length used to represent and process infinite-precision data. Because of the versatility of the board, students can study the influence of different finite-precision and floating-point representations on the performance of the control system.

THE ADVANCED EDUCATIONAL MICROPROCESSOR System combines a variety of attributes into a single educational tool. Teachers can use this tool to teach up-to-date techniques and concepts in computer engineering, electrical engineering, and other disciplines in which advanced microprocessor technology is applicable. In addition to a high-performance computing engine, the features that make the board attractive for educational purposes include easily accessible data lines and control points, user-controlled event counters, a configurable cache system, an Ethernet network interface, terminal access, and an easily utilized interface capability.

With these features the AEMS board is well suited to teaching situations in microprocessor systems, computer architecture, operating systems, and networks. The board also helps the teaching of computationally intensive applications and situations requiring both number-crunching and interfacing activities. This teaching area includes digital signal processing, controls, robotics, and communications applications.

Acknowledgment

The Engineering and Computer Engineering Department of the University of New Mexico received a Motorola Partnerships in Research and Development Grant.

References

1. A. Clements, *Microprocessor Systems Design: 68000 Hardware, Software, and Interfacing*, PWS-Kent, Boston, Mass., 1987.
2. T.L. Harman and B. Lawson, *The Motorola 68000 Microprocessor Family*, Prentice Hall, Englewood Cliffs, N.J., 1985.
3. W.A. Triebel and A. Singh, *The 68000 Microprocessor: Architecture, Software, and Interfacing Techniques*, Prentice Hall, 1986.
4. A. Wilcox, *68000 Microcomputer Systems*, Prentice Hall, 1987.
5. Motorola, *M68000 Tutor Program Assembly Listing*, 3rd ed., Motorola Inc., June 1982.
6. T.L. Harman, *The Motorola MC68020 and MC68030 Microprocessors*, Prentice Hall, 1989.
7. Motorola, *MC68030: Enhanced 32-Bit Microprocessor User's Manual*, Prentice Hall, 1990.
8. Motorola, *MC68881/MC68882: Floating-Point Coprocessor User's Manual*, Prentice Hall, 1989.
9. L.H. Pollard, *Computer Design and Architecture*, Prentice Hall, 1990.
10. L.H. Pollard, *The Design Book: Techniques and Solutions for Digital Computer Systems*, Prentice Hall, 1990.
11. K.J. Astrom and B. Wittenmark, *Computer Controlled Systems: Theory and Design*, Prentice-Hall, 1984.
12. A.V. Oppenheim and R.W. Schafer, *Digital Signal Processing*, Prentice Hall, 1975.

L. Howard Pollard is an assistant professor in the Department of Electrical and Computer Engineering at the University of New Mexico in Albuquerque. His research interests include computer design, parallel computing, distributed processing, and parallel computer architectures and applications. Prior to his work in academia, he was a research engineer for Lockheed Missiles and Space Company and a design engineer for Reticon Corporation.

Pollard received the BS and MS degrees from Utah State University in Logan and the PhD degree from the University of Illinois in Urbana.

Ramiro Jordan is also an assistant professor in the Department of Electrical and Computer Engineering at the University of New Mexico. His research interests include multidimensional signal processing, computer networks, and microprocessor systems design and applications.

Jordan received the BS degree from the Universidad Nacional de La Plata in Argentina and the MS and PhD degrees in electrical and computer engineering from Kansas State University in Manhattan.

Direct questions concerning this article to L. Howard Pollard, E&CE Dept., University of New Mexico, Room 110, Albuquerque, NM 87131.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 162 Medium 163 High 164

Hardware, multitasking

continued from p. 33

Peripheral modules...

problems the students discover is the 120-Hz noise produced by the fluorescent lights and the problem of selecting the best direction for tracking from one photocell. The usual project has become a bright-spot-detector and tracker that in December students call a "Rudolph tracker."

14) Under low-level development for several years but nearing completion is a DC gear motor with an incremental shaft encoder. It incorporates an inexpensive 12V gear motor with a small, slotted aluminum disk. Three reflective optosensors arranged around the disk provide quadrature signals and a mark channel. After several attempts at using logic arrays, we designed the current version to use a one-chip microprocessor for signal processing. It provides an absolute position code along with up and down pulses. We plan switch options to produce a serial-interfaced, smart motor controller with some sort of communication protocol as well, but we've yet to begin the software effort.

15) Various commercial devices including a "minimover" manipulator, an *x-y* digitizing pad, a barcode reader, and a parallel printer are available. We have eight manipulators and other commercial single-unit devices, but only a very few projects make use of any but the first devices.

With only 10 system calls, the DCX operating system is ideal for a first systems course. While it is much simpler than many operating systems, it shows the important multitasking ideas: task states, priority, synchronization, and message passing. With multitasking, individual tasks (call them program pieces, functions, or procedures) handle parts of the total job. For example, one task scans a keypad for user inputs. Between scans, another task writes to a display. A third, higher priority task issues pulses to a stepper motor at a regular interval. One microprocessor handles all these activities, and the operating system switches between tasks as necessary. The programmer writes each task as though the other tasks are of no concern; if something more important needs to be going on, the operating system manages it.

The operating system formalizes and simplifies details associated with interaction between tasks. Large, software-intensive systems can use multitasking, but the impact of task interaction shows up more strikingly in the lab when using "raw," unbuffered hardware such as motors, displays, and buttons. With the DCX operating system, overhead is small, and the system is easier to use. The hardware cost is low, yet the system can show the same ideas as in the nucleus or kernel of big-system multitasking systems.

Programming languages. We have used Intel's PL/M for programming. Because the embedded control field migrated to the C high-level language, our course is moving too. Several difficulties arose, however. Intel lacks a C compiler for the 8051 controller, and any C interface to the DCX operating system must be defined. It is possible to write some assembly-language hooks to DCX since the firmware has well-defined locations and parameter-passing conventions. C is usually quite stack-oriented and apt to produce large code modules. Once DCX is added to the 8044, very little on-chip stack space remains. Several 8051 C compilers can produce code that avoids stack-based operations. Such nonstandard C might be frowned on by a purist, but programs for the 8051 are highly hardware dependent. The 8051 was never designed to be a general-purpose computer, though it does run control jobs efficiently. Using even a nonstandard form of C still allows students to continue to program in the language they understand.

The course sequence. The course builds on a background in assembly language, microcomputer hardware, and a high-level language. From the start we place the emphasis on the multitasking ideas and the mechanics of the development process for the particular lab hardware. The first lab project involves running a program to turn a stepper motor. Students must use the system call RQWAIT to handle the timing between steps. By using the system's Bitbus monitor, one can change the value of the variable that sets the delay between steps and thus change the speed of rotation. A follow-on lab requires the individual student to make minor alterations to the program to change the direction of rotation (and thus go through the development process again).

After these get-acquainted labs, students form groups to develop a "multiaxis drilling machine" using four stepper motors. This project gives them their first taste of accomplishing more than one task at a time and illustrates the simplicity of adding more tasks without complicating the programming. It also introduces message passing between tasks when the user input must be translated and forwarded to the individual motor tasks.

A third lab assignment involves making a friendly teaching system for a stepper-driven robotic arm. While not conceptually different from the earlier lab, it gives the groups a chance to prove that they can do a better job of planning the second time around.

The remaining six to seven weeks involve new groupings to tackle a big project. Among the project suggestions are a motor-speed control system using a pulse output as a tachometer and an on/off control to produce a pulse width modulated drive signal. Students may choose to close the loop with a PID algorithm. A second choice is a scanning

photocell driven by two stepper motors in a polar fashion. With careful programming, the system can track a bright light source as it moves across a "sky." Other choices include control of the robotic arm, use of a music keyboard and programmable sound generator to make a synthesizer, remote control of a process control tank that is part of the separate controls lab, temperature control with a hair dryer and a temperature detector, and some sort of arcade game system.

In all the projects the goal is to have separable input, output, and combined phases to the project so that intermediate results can be shown. The students see the process of system integration as development progresses. Most of the hardware for the projects is available in prebuilt modules in which the students build the wiring from the ports to the modules. If they were required to build and debug the hardware involved, we would not have enough time to explore the multitasking ideas the projects illustrate.

For example, just getting the mechanical combination of a stepper motor and position sensor to work well could take a week, yet it clearly shows the impact of priority. When another equal-priority task takes too long, the pulses to the stepper are delayed and the motor shows a "glitch." This problem would not be so apparent if the hardware were only some lights and switches. In the same way, if a task driving a speech module does not sleep while a phrase is being uttered, everything else will stop while the module "talks."

The value of group projects. Students work in groups of three and divide projects into parts to be developed individually. Many students have never worked in groups with a technical goal (as opposed to communication courses in which the goal is a report or a presentation). They find it difficult to decide how to share the parts of the project. Many stronger programmers have never developed pieces of a program without being responsible for the total result.

Groups commonly find that the program pieces are not all the same size or difficulty. Even worse, we commonly find that not every member of the group understood the interactions between pieces in the same way. This finding becomes apparent as they integrate the pieces into a total system. Somehow students don't really consider the process of testing individual pieces before attempting actual projects and then beginning to put the pieces together. Four years of experience with group multitasking projects taught us three major lessons, as listed in the box.

The laboratory equipment. The laboratory uses Intel's Bitbus network based on an 8051-type processor (technically it is an 8044 that is an 8051 with the UART replaced with a SDLC controller). Commercial 100 × 220-mm Eurocards contain the controllers. Each microcontroller has its own multitasking operating system in ROM (Intel's DCX). We have many kinds of boards available. The 44/10 card shown in Figure 1 provides three parallel (digital) ports. The hardware can be hosted off PC equipment through an additional card. A monitor program allows the user to examine variables and address I/O from the host PC while the tasks are running on the remote board. It should be possible, for an additional cost of about $1,000 (beyond the cost of the PCs), to assemble a station with two or three nodes. If development is to be done in the C language, a compiler for the 8051 family may entail a significant cost. At least six versions are on the market, and pricing policies are not clearly established for academic institutions. Running under DOS lets us make BAT (batch command) files that insulate the students from many specific software development details that are irrelevant to the ideas involved.

Tips for group projects

1) **Plan big but start with very small pieces.** At the beginning of the semester students attempt to include all the "bells and whistles" and expect everything to work. By the end of the semester they say, "First let's just get a character to show on the LCD. Then we'll send one message. Then we'll implement the user prompts ..."
2) **Test the hardware before you test the software.** As in many other microprocessor courses, students tend to suspect one's software and implicitly trust the hardware provided in the lab. Multiple software revisions and total rewrites occur even though the problem was defective hardware or defective understanding of how the hardware responds to signals from the computer. By the end of the semester the students come to appreciate the confidence generated by the firm knowledge that the external hardware responds at the specified address in the specified manner because they have run simple test routines. When the computer tells the stepper motor to take a step in the defined manner, they are confident that it will happen. If it does not, they then know the failing lies with some recent revision to the software.
3) **Someone will get the message format messed up.** Students working in groups don't listen to each other. While they are planning the messages to pass between tasks, everyone nods and agrees about the order of the various pieces of information to exchange and what to do with them, but apparently not everyone agrees the way others thought they agreed. Frustrating as it is to the students, they come to have a high appreciation for software specifications. The personal experience provides a powerful motivation to change. By the second or third project, most groups spend several hours outside class hammering out what is to be done and who is to do it.

WE'VE FOUND THAT THE VARIETY of devices that can be "experienced" by the students is much greater than would otherwise be possible in the brief lab times associated with most microprocessor courses. When compared with most commercial equipment associated with computers and electronic design, this hardware is inexpensive. We will provide more detailed design information to interested educators upon request.

The multitasking course has induced a level of project-development sophistication that is not present when students have taken only the earlier courses. Besides the modular-programming and multitasking ideas, students get a practical appreciation for the importance of good planning and coordination among group members. The hardware itself has proven to be practical and affordable for developing embedded control applications. μ

References

1. *Guide to Using Distributed Control Modules,* Intel Corp., Santa Clara, Calif., 1984.
2. "Inside a Real-Time Kernel," *Embedded Systems J.*, Vol. 3, No. 11, Nov. 1990, pp. 35-47.

Thomas W. Schultz, associate professor of Electrical Engineering Technology at Purdue University, West Lafayette, Indiana, has taught microcomputer courses for 10 years. Before coming to the university, he worked in design and development for GTE Sylvania and Bell Telephone Laboratories. His main interest is in embedded-control applications of microprocessors.

Schultz received the BS degree in electrical engineering from the University of Connecticut and the MSEE degree from the University of Illinois. He is a Senior Member of the IEEE.

Address questions concerning this article to the author, Purdue University, Electrical Engineering Technology, Knoy Hall of Technology, West Lafayette, IN 47907.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 168 Medium 169 High 170

Course sequences

continued from p. 37

level interface programs is a good first introduction to the concept of a device driver.

After basic interfacing, students analyze the motherboard circuitry of a more complex microcomputer such as a PS/2. A good approach starts with basic bus operations such as DMA, and then moves on to DRAM interfacing and an introduction to coprocessor interfacing.

The final section of the first-term class covers CRT and disk interfacing. The hardware part of this section includes discussions of the programmable very large-scale integration (VLSI) devices now commonly used to interface with these devices. Since CE and EE students have usually taken at least one C or Pascal programming class, a separate section on C programming is not needed. We cover the main programming skills here—assembly language drivers and interfacing assembly language modules with, for example, C mainline programs.

Second term. This class starts with a description of the design project. A 386-based digital oscilloscope or a 386-based logic analyzer are examples of student projects that bring many hardware and software pieces together. Although it is possible to build the prototype of the entire project from scratch, we use a preassembled board such as the URDA 80386 μLab for the microprocessor core of the project to make it easier to complete the project in one term.

The next class section involves a discussion of simulation and other system design techniques. Ideally students learned schematic capture and simulation in their lower division logic design classes. But in any case, it is now a good time for the instructor to show how to use simulation to check the logic and timing of a microcomputer design. Part of this instruction might include the reasons why the wire-wrapped prototype of a 50-MHz microcomputer will probably work very differently from a version built on a multilayer PC board. The instructor might also mention that Apollo/Hewlett-Packard reportedly used simulation to both shave several months off the time to market and drastically reduce the number of design spins for one of their current engineering workstations.

Incidentally, Mentor Graphics will donate their schematic capture, simulator, and other tools to universities for a small, yearly upgrade charge with the acquisition of the appropriate workstations. When working with the Mentor tools, I recommend that instructors also acquire the appropriate Smartmodels from Logic Automation. These Smartmodels give very detailed messages if any logic or timing problems are found during a simulation run. For the PC-based environment, the Capfast schematic capture program from Phase Three Logic and the Standard Universal Simulator for Improved Engineering (SUSIE) from Aldec are good tools. Both

of these companies have generous pricing for universities.

Once the project has been defined, the next step is to cover the 386 hardware characteristics so the students can get to work on the core of the project. The instructor can deemphasize the 286, because most new designs will probably use a 386SX, 386DX, or i486. The 386 hardware discussion leads into an introduction to static RAM caches, and error detecting/correcting techniques and devices.

Most of the protected-mode features of the 386 and i486 processors don't make much sense unless students understand the basic needs of a multitasking operating system. Therefore, we briefly discuss these needs next. With this background, students seem to understand the protected-mode features of the 386 and 486 with little difficulty. Again, the 80286 can be deemphasized because of its difficulty in switching back and forth between real and protected mode.

If the core of the design project involves the URDA 80386 µLab board, students can develop the project software on a PC and simply download the code to the µLab board through an RS-232C cable. The board's debugging monitor program operates the 386 in protected mode using the flat memory model, the same memory model used by Microsoft's Windows 3.0 and OS/2 programs.

The advantage of the flat memory model is that it effectively eliminates segmentation because all segments start at absolute address 00000000 and extend to the full 32-bit address space if needed. Since simple 16- or 32-bit offsets access code and data words in this single shared segment, students find it very easy to write application programs for the flat model.

During initialization the µLab monitor program sets up a system task for itself and a user task for user programs. Students can write software for the design project with Borland, Microsoft, or Intel tools on a PC- or PS/2-type computer. They can download the software to RAM on the board at any address above 300H. The monitor run command switches execution to the user task.

The software resident on the µLab board allows students to concentrate initially on the software that actually controls their digital scope or logic analyzer, instead of concentrating on the software that sets up the 386 descriptor tables. When the application part of the program is working correctly, students can modify the software to work in a stand-alone manner before it is burned into erasable, programmable read-only memory (EPROM) and tested on the board. This PC-µLab board approach is a relatively inexpensive alternative to the full-fledged development system and emulator approach. If you are fortunate enough to have an emulator, you can just plug the emulator into the µLab board and use it to download, test, and debug the software.

To develop the hardware for the core part of the 386 system, the best approach is to simulate the design using, for example, the Mentor-Logic Automation tools. When the design passes simulation, students make a board and then test it using an emulator and/or a logic analyzer.

This second microprocessor design class finishes with short introductions to reduced instruction-set computing (RISC) processors, parallel processors, neural networks, fuzzy logic, and other current topics of interest. These discussions are mostly for the benefit of the EE students in the class, because CE students will most likely have covered these topics in a previous or concurrent computer architecture class.

MY COLLEAGUES AND I HAVE FOUND that adaption of a common core of microprocessor curriculum materials works well for students with a range of skill levels in different course sequences. By customizing course materials, we can maintain a high level of student motivation and help them to develop strong skill sets.

Douglas V. Hall is an independent consultant, author, and adjunct instructor in the Electrical Engineering Department at Portland State University. He previously taught microprocessor classes at Portland Community College. He has published several books on microprocessors and digital systems.

Hall received a BS in physics from State University of New York, Albany and is working on a PhD in electrical and computer engineering at Portland State University.

Address questions concerning this article to Douglas Hall, Department of Electrical Engineering, Portland State University, PO Box 751, Portland, OR 97207-0751, or on Internet at dough@ee.pdx.edu.

Bibliography

D.V. Hall, *Experiments in Microprocessors and Interfacing,* McGraw-Hill, Westerville, Ohio, 1987, 2nd ed., to be published in 1992.

D.V. Hall, *Microprocessors and Interfacing, Programming, and Hardware,* McGraw-Hill, 1986, 2nd ed., to be published in 1992.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 171 Medium 172 High 173

Futurebus interface

continued from p. 41

register the release of AR* and then move to the next state.

4) When an arbiter is ready to move on again, it asserts aq* and releases ap*.
5) As before, only when all arbiters are ready to move will all boards release ap* and then bus signal AP*.
6) Subsequent state transitions repeat a similar process.

A minimum of three wires are required to achieve this rather ingenious synchronization protocol. The main arbitration sequence actually has six states, which require two complete cycles of the three-wire synchronization protocol.

We are now ready to sketch the sequence of actions required to achieve bus mastership. Since this activity uses different bus signals than the data transfer protocol, it can take place concurrently with data transfer transactions driven by the current master.

1) Any arbiter requiring mastership initiates the protocol by asserting ap* and releasing ar*. All other arbiters accept the assertion of AP* and signal their willingness to proceed by also asserting ap* and releasing ar*. At the same time, they indicate whether they too will compete for mastership.
2) Each competing board applies its unique arbitration number an[7 ... 0]* to the bus signals AB[7 ... 0]* in the following way. It asserts ab[n] if and only if:

   - an[n] is asserted, *and*
   - for all $m > n$ when an[n]* is released, so is AB[m]. This condition fails when some higher priority board drives the bus, thereby asserting AB[m] for some m. In this case, all lower priority boards release ab[$m - 1$ ... 0]*, so that they do not interfere with the lower order bits of the high-priority board's arbitration number.

     The Futurebus standard calls the combinatorial network implementing this scheme the arbitration ladder. Effectively, after a certain time delay, the signals on AB[7 ... 0]* will be the largest arbitration number of any competing board.
3) Each arbiter times a delay appropriate to its own arbitration ladder and signals its willingness to proceed using the AP*/AQ*/AR* protocol.

Transferring a 5-word packet

Transfer of this packet occurs in the following sequence of events.

1) The sender gains control of the bus, using the arbitration protocol described in the Asynchronous Design section. The sender therby becomes the bus master.
2) The master places the address of the recipient on the address/data signals AD[0-31]* and then asserts the control strobe as*. The strobe, in turn, causes the bus signal AS* to be asserted.
3) When AS* is asserted, *each and every* board decodes the address. When decoding is complete, the board records whether it is being addressed, asserts the acknowledge signal ak*, and releases a complementary signal ai*.

   Each individual board always drives the two signals ak* and ai*. Notice the overlap between the assertion of the bus signals AK* and AI* inversely. When the fastest slave completes address decoding, it asserts ak* and hence AK*. It also releases ai*, but AI* remains asserted because other slower slaves are still asserting it. When the slowest slave completes decoding, it releases ai* and hence AI* will finally release, signaling the end of the address beat.
4) When AI* releases, the master can safely remove the address from the bus. It cannot do so earlier, because some slaves may still be decoding the address and will become confused if the data is removed.
5) Next, the master places the first word to be transmitted on the AD[0-31] bus and asserts the data strobe ds*.
6) When DS* asserts, the recipient reads the data from the bus. When it has successfully captured the data, it asserts dk* and releases di* to indicate this fact.
7) When DI* releases, the transfer of the first data word completes, signaling the end of the first data beat. The master then places the next data word on the bus and releases ds*. Since only the master drives ds*, DS* will be released too.
8) When DS* releases, the recipient grabs the next data word from the bus, and when the data is safely captured, it asserts di* and releases dk*.
9) Data transfer now continues in this way, one word being transferred for each transition of DS* and DK*/DI*, until the master has no more to send. Then the master releases as* (which has been asserted all this time).
10) The recipient, and all other slaves, acknowledge the release of AS* by releasing ak* and asserting ai*.
11) When AK* releases, the transaction completes.

4) When all arbiters are willing to proceed, the signals on AB[7 ... 0]* have stabilized. The winning board, whose arbitration number is on the AB[7 ... 0]* bus, now becomes master elect.
5) When the current master completes its transaction, it permits the arbitration sequence to complete (using AP*/AP*/AR* as before). The master elect becomes the new current master.

As described, this sequence would always elect the arbiter with the highest arbitration number, but Futurebus superimposes another protocol on top to allow fair access. Once an arbiter has become master, it becomes a withholder and refrains from competing for the bus until only withholders remain. Discovering this fact requires an extra bus signal AC*, which is asserted by any board wishing to compete. An arbitration sequence in which no participant asserts AC* unlocks the withholders, which are then free to compete again.

The complete Futurebus+ protocol is quite a bit more complicated than this. It allows preemption of the master elect, arbitration messages, parity checking on the arbitration number and error recovery, a two-cycle arbitration to allow larger arbitration numbers, a fast-track method for gaining access to an idle bus, and so on. Futurebus+ also substantially improves the fairness protocol just described to allow round-robin access within priority levels.

Wired-Or glitch. One awkward feature of using an open-collector bus to support multiparty transactions is the wired-Or glitch. If several drivers are asserting a bus signal and one releases it, the others must sink more current to compensate. Unfortunately, they do not register this fact right away; instead, a wave propagates from the released driver outward along the bus in both directions. This action causes intervening receivers to accept the signal momentarily released, before the other asserting drivers start to sink the extra current. The glitch can last for at most two bus lengths, or about 25 ns for a Futurebus. Interposing a glitch filter in the receiver of every bus signal used in a wired-Or manner solves this fundamental problem.[1]

Glitch filters impose an unavoidable delay between the release of a bus signal and the time this event can be acted upon. This constraint limits the maximum data transfer rate in compelled-mode broadcast operations. When only one recipient is involved, glitches cannot occur and speed increases upon bypassing the glitch filters on DK* and DI*.

Asynchronous design

Both the master and the slave need to incorporate some sort of finite-state machine to manage the data transfer protocol. Conventionally, these finite-state machines would be synchronous, an approach with some serious disadvantages for this application. The main innovative feature of our interface is that its state machines are entirely asynchronous.

Disadvantages of a synchronous interface. A conventional synchronous design for a Futurebus interface suffers from certain problems. We refer to Figure 5, a block diagram of a typical synchronous state machine.

- A synchronous system must take discrete samples of control signals coming from the bus so that the signals are stable while the finite-state machines make their transitions (latch A in Figure 5). This step results in an average half-clock-cycle delay between the transition on a control signal and its effect on the finite-state machine.
- In our application a further full clock-cycle delay would be required before any output can be generated in response to the input transition because the outputs themselves must be latched (latch B in Figure 5). The outputs cannot be derived combinatorially from the sampled inputs, because the decoding circuitry would cause race hazards on the outputs. Such a condition is not tolerable when these outputs are driving Futurebus control signals, which must be free of glitches.
- Worse still, every time a clocked register samples an asynchronous control input, a danger of metastability failure exists.[8,9] Metastability occurs when the data input of a clocked element makes a transition at a time sufficiently close to the clock transition; the output of the element can then hang in an intermediate state for an unbounded length of time.

We cannot escape metastability failure. However, its probability decreases exponentially as the time increases between the clock transition and the time at which the output is required to be stable. For current TTL (transistor-transistor logic) technology, for example, Fairchild's Advanced Schottky TTL,

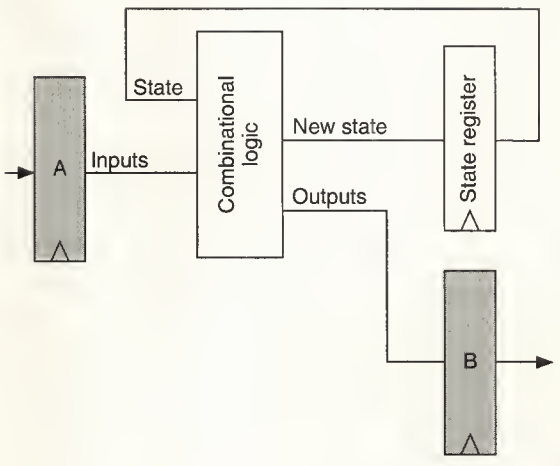


Figure 5. Synchronous state machine.

a reasonable settling time is around 40 ns. This settling time restricts the use of TTL samplers to signals with a duty cycle of considerably more than 40 ns.

The first two factors impose a minimum one-and-a-half clock-cycle delay between an incoming strobe and an outgoing response. The third factor limits the extent to which we can increase clock speed in compensation for this delay.

Asynchronous FSMs. An alternative design method uses asynchronous finite-state machines in the Futurebus interface. These machines make state transitions in direct response to transitions on their inputs. Of course, the inputs to asynchronous FSMs must therefore be free of glitches, but this is in any case a fundamental requirement of the entire Futurebus protocol. Figure 6 shows the basic structure of an asynchronous FSM for comparison with the synchronous machine given in Figure 5. The simplest example of an asynchronous FSM is a set-reset latch.

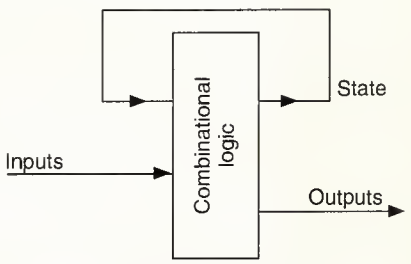


Figure 6. Asynchronous state machine.

Such a design suffers from none of the problems mentioned for synchronous machines. It provides no input synchronization latch nor output latch, so only combinatorial delays from inputs generate outputs. Furthermore, since we have no synchronization latch, no risk of metastability exists. Asynchronous FSMs therefore operate both faster and with a lower risk of error than synchronous interfaces. We discuss later why the risk of metastability cannot be eliminated altogether.

Asynchronous FSMs can be implemented readily using off-the-shelf combinatorial PALs (programmable array logic) chips. These devices invariably (and conveniently) contain feedback terms, which are required for asynchronous FSMs. Note that it is more difficult to generate the combinatorial logic for asynchronous FSMs than for their synchronous counterparts, since every transition function must be free of hazards.

While a large number of computer-aided design tools exist for synchronous FSMs, none exist for asynchronous machines. Worse, while we have good literature on asynchronous, or self-timed, systems,[10] few papers purport to give practical design techniques. As a result, we developed a new method for generating asynchronous FSMs from their specification and implemented this method in a computer program. After that,

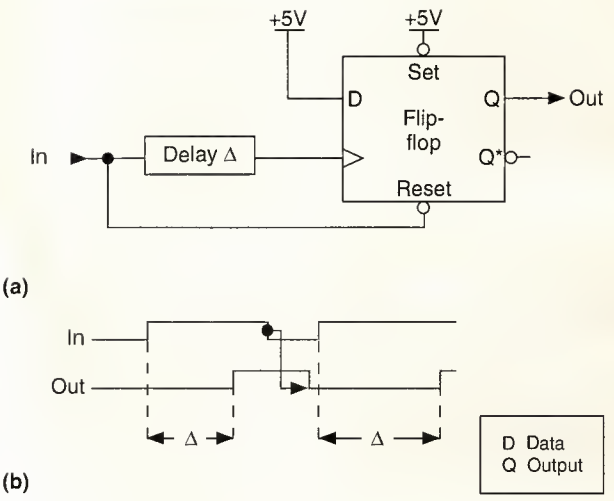


Figure 7. Resettable delay line circuit (a) and timing (b).

the main obstacle was that the program often generated terms that were too large to fit into commonly available PALs. Similar situations arise with synchronous FSMs, and the pragmatic solution is the same. We must alter either the state assignment or the machine specification to reduce the number of product terms.

Peyton Jones[11] fully documents our design method, Sutherland[12] offers an excellent modern introduction to asynchronous systems, while Fletcher[8] gives a fuller treatment.

Time delays in asynchronous systems. Often we must time a delay in an asynchronous system. For example, during the Futurebus arbitration protocol, the priority resolution circuitry requires a certain time to settle. A synchronous system would time such delays by counting clock cycles, but an asynchronous system requires delay lines. Our implementation of the Futurebus interface required four delay lines.

Delay lines are inadequate in one particular respect. Typically, an asynchronous FSM will time a delay ΔT by asserting a signal that drives the delay line and waiting until the delay line output becomes asserted. At this point the FSM releases the delay line input. Unfortunately, the output of the delay line remains asserted for a second ΔT, the recovery period. The process cannot be repeated during the recovery period, because the FSM will erroneously interpret assertion from the first delay as indicating the completion of the second delay.

This is a serious problem because the Futurebus protocol contains a number of instances of a fairly long delay that may be rapidly repeated. The arbitration delay previously mentioned is one example; another is the address decoding delay triggered by AS*. Furthermore, since the FSM must await the release of the delay line output before reasserting its input, the complexity of the FSM increases.

What is required is a sort of delay line that resets completely when its input is released. We modeled this with the circuit shown in Figure 7a, b. We use the output of the delay line to clock an edge-triggered latch, whose data input is always asserted. The reset input of the latch connects to the input of the delay line. The latch output is therefore reset as soon as the input is released. It will not be asserted again until the transition caused by the assertion of the input makes its way through the delay line.

Futurebus interface

The Futurebus interface divides into four almost entirely independent parts (see Figure 8):

- the data path,
- the Futurebus Receive machine,
- the Futurebus Send machine, and
- the Futurebus arbiter.

BIP regards the Send and Receive machines as completely separate masters of the local bus.

Operation. We use only one kind of bus transaction (block writes with a two-edge handshake, as described earlier). We breached the Futurebus standard by omitting all use of the CM* command signals and drastically restricting our use of the ST* status signals.

Address beat. The Send machine on the sending board transmits the first word of a packet during the address beat, which begins when the master asserts AS*. Each Futurebus interface decodes a 5-bit board-address field from AD[31 ... 27]* during this beat. Decoding determines whether the packet is destined for the Futurebus board, sets a status indication appropriately, asserts ak*, and releases ai*.

The status indication during the address beat is a nonstandard variant of the Futurebus protocol. Say a board recognizes the address as its own, and it has a free packet frame to receive the packet. It then asserts the sl* status signal. All unselected boards, or boards without free packet frames, release sl*. If the SL* status signal is released, it follows that either no board is selected or the selected board has no room for the packet. In either case, the transmitting board queues the packet for later retransmission.

This artifice, whereby a full board pretends to be unselected, allows us to use one status signal that, as luck would have it, saved us a whole transceiver package. Packets sent to nonexistent boards accumulate conveniently in the Send Queue. Here they can be discovered by the diagnostics system.

The address beat is all that takes place for a one-word packet—an example of the usefulness of the Futurebus address-only transaction.

Data beats. The second and subsequent words of each

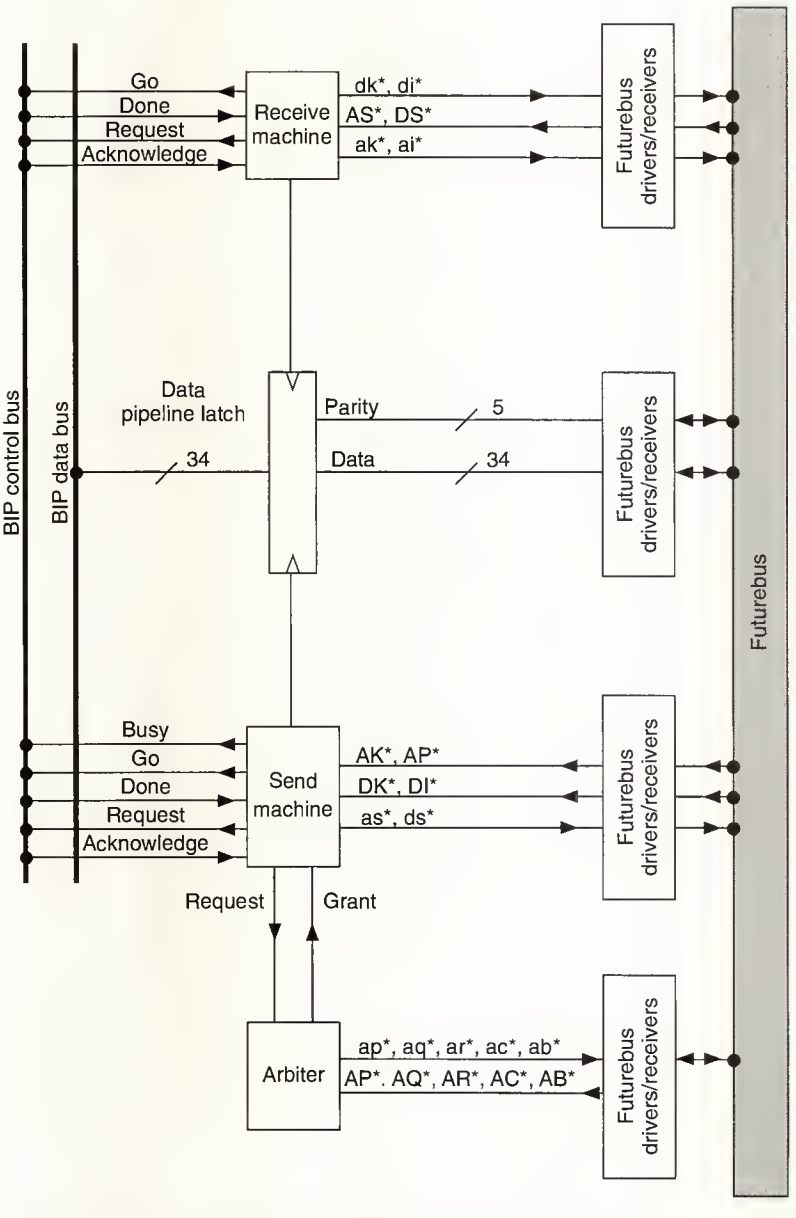


Figure 8. Block diagram of Futurebus interface.

packet move across the Futurebus using the two-edge-handshake, block-transfer protocol.

All data words act as broadcast cycles using the complementary DK*/DI* data acknowledge signals, to allow a diagnostic board to monitor data transfer (see Diagnostics section). Replacing the glitch filters on DK* and DI* with a wire link would speed up data transfer but would sacrifice the capability to monitor such transfers.

Receive actually consists of no less than three coupled, asynchronous FSMs.

In principle, only 33 data bits need be transferred in a data beat, because the Receive state machine can regenerate the 34th bit, indicating the last word in a packet. Unfortunately, this method seems to require an inevitable slowdown, for the following reason.

Before the data can be written into the buffer RAM, the Receive machine must acknowledge that data as the last word or not. It can only do so by registering whether the next control transition is a change of state on DS* (indicating that a further data word is being transmitted) or the release of AS* (indicating that no further data words follow).

So when the next transition on DS* occurs, the Receive machine must first write the previous word, now complete with its last-word indicator bit, into the buffer RAM before it can strobe data from the bus into the pipeline latch. This is clearly a waste of time because buffer-RAM writes do not overlap with bus transfers.

The correct solution provides a second pipeline latch. But this requires considerably more hardware and significantly complicates the Receive machine. We chose instead to use one command bit in a nonstandard way to indicate that the current word is the last of a transaction, in either an address beat or a data beat.

Data path. The data path is quite straightforward. A bidirectional pipeline latch interfaces between the Futurebus transceivers and the buffer-RAM data bus. (Latching Futurebus transceivers were not available.) In addition to its bidirectional latching capability, this pipeline latch generates and checks parity. We used F552s for this purpose; they are available only from Fairchild in large, 0.6-inch ceramic packages.

The 33rd and 34th data bits and their parity bits were tiresome to work with, because they didn't fit neatly into 8-bit packaging.

As the address word is written from the pipeline latch into the buffer RAM, the board address of the current master replaces the 5-bit board-address field. This field redundantly contained the address of the receiving board, so this replacement neatly informs the recipient of the sender's board address of the sender.

Receive machine. The Receive section accepts the Futurebus AS* and DS* address and data strobes. It also drives the ak*/ai* and dk*/di* address and data acknowledge signals.

When it decodes the address word of a packet destined for its board, Receive acquires mastership of BIP. It then strobes successive data words into the pipeline latch from the bus and writes them into the buffer RAM, with these latter two operations being overlapped. While doing so, Receive checks the parity of the received words. When a complete packet has been written into the buffer RAM, Receive relinquishes BIP mastership.

The Futurebus should be free for further traffic as soon as the last data word is written into the pipeline latch. In the special case of one-word (address-only) packets, this means that the Futurebus transaction is not even required to wait for Receive to gain mastership of BIP.

This final point is not as trivial as it sounds. As described earlier, *all* boards participate in the address beat. So, all boards have signalled completion of address decoding before the master starts to transmit data. Hence, while Receive writes the final word of a packet into the buffer RAM, it should simultaneously be capable of responding to an arbitrary number of address cycles to other boards. (Of course, when a further address cycle is directed to a board that is still clearing its pipeline latch, the board must delay its acknowledgment until the pipeline latch has been emptied.)

Consequently, Receive actually consists of no less than three coupled, asynchronous FSMs. One FSM autonomously responds to AS* by driving ak*/ai* appropriately. This FSM only has two states. A delay line restrains AS* before it moves to this FSM to allow time for the address decoder to complete.

The second FSM manages the DS*/dk*/di* Futurebus interaction, acquisition of BIP mastership, and data transfer into buffer RAM. This FSM has six states.

The third FSM "remembers" whether the Futurebus transaction being processed by the second FSM has been completed, that is, whether AS* has been released. Since further transactions might now be taking place, the state of AS* is not a reliable guide. Also, the second FSM might still be in the process of writing data into the buffer RAM. As a result, a separate piece of state retains the information that the Futurebus transaction has been completed.

Send machine. The Send section consists of one asynchronouus FSM. It is the most complex single asynchronous FSM in the Futurebus interface, with 16 states. Whenever the Send Queue is occupied, Send requests Futurebus mastership from the Futurebus arbiter. When it is granted Futurebus mastership, Send requests mastership of BIP. It

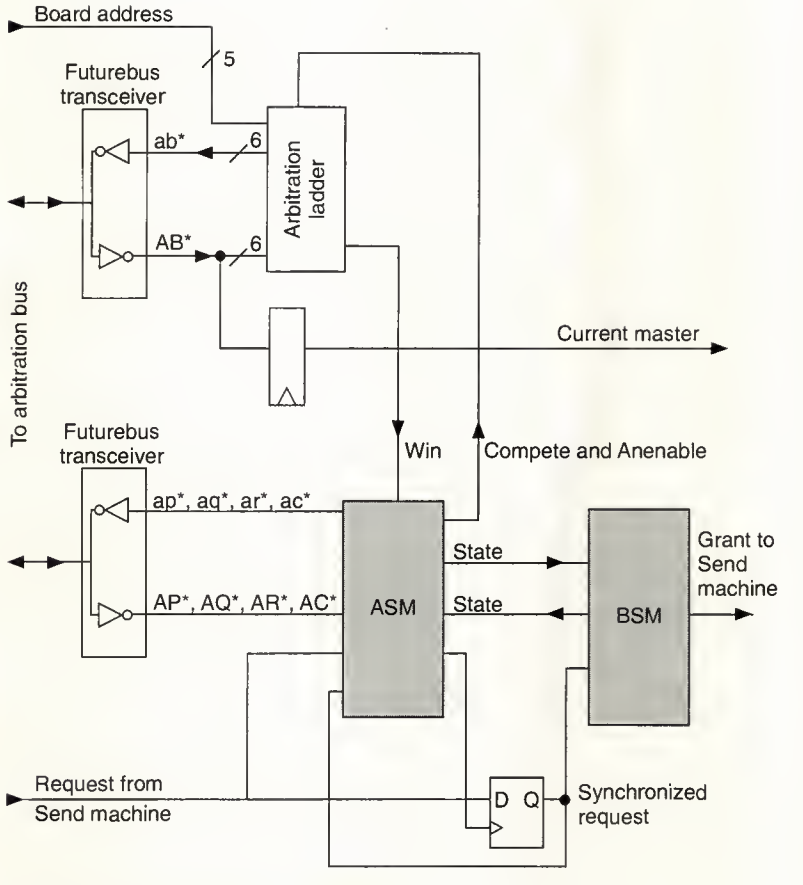


Figure 9. Futurebus arbiter.

cannot make this request before acquiring Futurebus mastership, because deadlock could result if Send acquired BIP mastership while another Futurebus master was trying to send packets to that board. If Send held BIP mastership, Receive would hold up the Futurebus indefinitely while it tried to acquire BIP mastership. Simultaneously, Send would retain BIP mastership indefinitely while it tried to acquire Futurebus mastership.

Upon acquiring BIP, Send begins sending packets over the Futurebus, using the as* and ds* Futurebus control strobes. Send fetches data from the buffer RAM simultaneously with the transmission of the previous word over the Futurebus.

When the destination board refuses a packet and the packet contains more than one word, Send does not transmit the word it has prefetched into the pipeline latch. Instead, Send must take another BIP cycle to fetch the first word of the next packet (if any). Simultaneously, the refused packet moves into the Resend Queue.

Futurebus arbiter. The arbiter consists of three parts, each implemented in one PAL chip (see Figure 9). A six-state arbitration state machine (ASM) manages the AP*/AQ*/AR*/AC* protocol on the arbitration bus. ASM also implements the state machine that sequences the arbitration protocol.

The board status machine (BSM) has three states: competitor, master, and withholder. This machine controls the output enables for Futurebus data transmitters and keeps track of whether or not the board may compete for mastership. Also in this PAL is a small two-state machine that manages the request/acknowledge handshake between the arbiter and Send.

The arbitration ladder network implements the priority scheme during competition, whereby the highest priority competing board becomes the next master. Two signals: Anenable and Compete control the ladder. When Anenable is released, the ladder unconditionally releases its ab[7 ... 0]*. When Anenable and Compete are asserted, the ladder implements the combinatorial arbitration network specified by Futurebus. When Anenable is asserted and Compete is released, the ladder unconditionally places the board's arbitration number on the bus. (This combination is used when the board has successfully acquired mastership.)

The arbitration ladder's output Win indicates whether it won the competition. AP* latches Win transparently, so that the output remains stable even when the unsuccessful boards start to remove their arbitration numbers. (Otherwise, glitches might occur on Win.)

The PAL we used was not quite large enough to include the most significant arbitration number bit, which is used for priority arbitration. Special-purpose silicon would probably fix this problem and increase speed somewhat.

We implemented only a subset of the full Futurebus arbitration protocol.

At exactly one place in the arbiter the risk of metastability cannot easily be avoided using standard parts. When a board initiates the arbitration cycle, all the other boards must compete for mastership, depending on whether their Send machine is requesting the access to Futurebus. They clock Send's Futurebus request into a latch and wait long enough to reduce the probability of metastability failure to an acceptable level.

In practice, the BSM PAL drives the synchronization latch input. This PAL only asserts the latch input when Send is requesting the Futurebus *and* the BSM is in the competitor

state. In this way the BSM can inhibit new Futurebus requests when it is in the withholder or master states. This process prevents the board from reacquiring Futurebus until an unlocking cycle has taken place.

Using a mutual-exclusion element could avoid this metastability and improve the typical-case speed. This element accepts two inputs and produces two outputs that indicate which input was asserted first. If the two inputs are asserted at the same time, the device may take an unbounded time to distinguish between them. But by having two outputs, it can indicate a decision (so the arbiter can proceed) as well as the type of decision made. The typical-case behavior (when the two inputs do not coincide in time) takes place very quickly. In addition, no possibility of failure exists because the element is not constrained to make a decision within a finite time. Unfortunately, such devices are not readily available to a board-level designer!

Diagnostics

The fully asynchronous nature of the Futurebus protocols allowed us to implement some unusually powerful and simple diagnostic tools. The ones of importance here concerned the Futurebus and BIP.

Futurebus diagnostics interface. We built a special board that interfaced the Futurebus signals directly to a Unix host computer. The host both reads the state of each Futurebus signal and asserts it under the control of a program running in the host. We then wrote a diagnostic program that displayed the state of the bus in an on-screen window and permitted us to observe bus transactions taking place one at a time.

In a fully asynchronous system, noninvasive diagnostics are simple to implement.

In the case of arbitration, for example, the program displays the states of AP*, AQ*, AR*, and AC* and acts as a (very slow) potential master. Using this system, we step through the arbitration cycle under interactive control, while separately observing the states of the arbitration FSMs on the boards under test. The diagnostic program can also request mastership itself. It implements a complete software version of the arbitration protocol, including the combinatorial arbitration ladder.

In a similar way, we may snoop on individual data transfers over the bus. For this reason the broadcast protocol performs broadcast data transfers, using complementary DK*/DI* acknowledge signals. Alternatively, the diagnostic program may act as the source or recipient of data transfers.

The diagnostic Futurebus interface provides four levels of participation in Futurebus activity:

- no participation at all;
- participation in arbitration but not in data transfers; the latter therefore run at full speed;
- participation in arbitration and address beats but not data beats; the latter run unimpeded; and
- full participation.

The interface moves freely between these levels, except that once it relinquishes participation in arbitration it cannot safely reinsert itself into the bus operations of a running system. Reinsertion corresponds directly to live insertion of a new board, and the Futurebus standard specifies some technology-dependent timing parameters in the live-insertion protocol. A software emulation of the protocol cannot possibly meet these timings. (We considered solving this problem by adding a full hardware arbiter to the diagnostic board but decided that the benefits did not justify the cost.)

BIP diagnostics. In a similar way, BIP contains a number of diagnostic ports, also interfaced to the host, which allow its internal state to be inspected. We added the diagnostic system as a perfectly standard, potential BIP bus master (it has highest priority). It must acquire BIP mastership before altering the BIP state.

Like the Futurebus interface, we designed BIP to be asynchronous, so it is amenable to snooping in the same way as Futurebus transactions. The diagnostic system controls a pair of control signals that hold up BIP's asynchronous handshake at a point late in the cycle. The diagnostics can thereby noninvasively trace all BIP operations by activating the one-step signals, waiting until a cycle reaches the point of being held up, and observing the state of BIP control and data signals.

We found this system extraordinarily useful for hardware debugging also. BIP can be held up in a state that is otherwise ephemeral but which is exactly the state in which hardware bugs are most apparent.

The diagnostic system also initializes the queues on system reset; for example, the free stack must be initialized with the addresses of all the packets in the system.

Summary. One of the major advantages of a fully asynchronous system must surely be that noninvasive diagnostics are so simple to implement. No modifications need to be made to the state machines that are being observed. All we need is an independent means of preventing the handshake from completing until a diagnostic system has observed the transaction.

The entire GRIP system relies on the diagnostic interface,

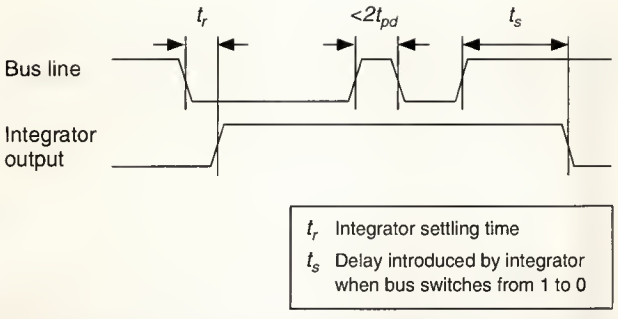


Figure 10. Integrator performance.

driven by a variety of host programs, for booting, initialization, and debugging. These programs provide an easy-to-use and easy-to-modify user interface, in contrast with the rather rigid interfaces normally provided by hardware front panels.

Lessons

The Futurebus standard describes a complex and sophisticated communications system. Implementing a small subset of the protocol cost us a substantial amount of board area, and clearly we would need custom VLSI silicon to implement the whole protocol.

We now review briefly the most painful aspects of our experiences, in the hope that others may avoid them.

Glitch filters. It would be particularly helpful if the Futurebus synchronization signal transceivers contained integrated glitch filters. Possibly these filters could provide filtered and unfiltered outputs (sometimes one is required and sometimes the other, even on one bus signal).

Glitch filters are particularly awkward to implement using discrete parts. Recall that the filters must eliminate all glitches, lasting less than 25 ns, from the asserted state to the released state. The operation is asymmetrical, however. As soon as the input returns to the asserted state, the filter must instantly reset, ready for another glitch (see Figure 10). As an extreme example, a waveform with alternating periods of 5-ns asserted and 24-ns released must be filtered to a constant asserted state. The requirement to filter glitches of up to 25 ns implies that the glitch-filter output will be delayed by a time t_s from the release of the bus signal, where t_s is at least 25 ns. The standard also requires that t_s be less than 45 ns whenever live insertion occurs.

Figure 11 shows our circuit, which consists of a resistor-capacitor RC network driving a Schmidt trigger gate. Notice the diode, which (imperfectly) implements the asymmetry. This circuit is unpleasant to work with for a number of reasons. The values of resistor R and capacitor C are likely to drift with time. Far worse, however, is the uncertainty of the trigger point for the output gate and the rate at which the input gate charges and discharges the capacitor. The net result is that it is impossible to design a predictable value of t_s, or to be sure of meeting the standard, even with hand-selected values for R and C. We chose values that ensured t_s would always be greater than 25 ns. But these values result in a worst case considerably greater than 45 ns, in turn resulting in a serious performance degradation of the overall system.

High-precision glitch filters might be implemented in silicon fairly easily, saving real estate and improving the speed of our implementation.

Skews. The Futurebus protocol is a model of rectitude, which admirably ensures the technology independence of the protocol. Unfortunately, it is a thorough nuisance to implement! We illustrate this with two examples.

Data/control skews. A fundamental tenet of the protocol requires that no strobe should appear on the bus before the data to which it relates is stable on the bus. Now, suppose a signal strobes data into a latch for transmission over the bus. It would be convenient to use that same strobe to drive the Futurebus control strobe (as* or ds*); but no, it must be delayed to allow time for the data to appear on the bus. At the receiving end, the incoming strobe must be delayed again before being used to clock the receiving latch to take into account the buffer skews. This requirement results in a proliferation of delay lines consuming real estate. Furthermore, since delay lines are only available in discrete steps that must be chosen to overestimate the actual delay required, more lines mean a greater overestimate and a slower system.

We breached the fine principles in this case, by using one delay, at the receiving end only, of AS* and DS* to absorb all the skews of the transmitting end and the receiving end together. This solution was only possible because we were designing a homogeneous system of identical boards.

Complementary acknowledge signals. The address and data acknowledge signals come in complementary pairs AK*/AI* and DK*/DI*, to allow broadcast and broadcall operations. Each board always drives ai* to the inverse of ak*, and simi-

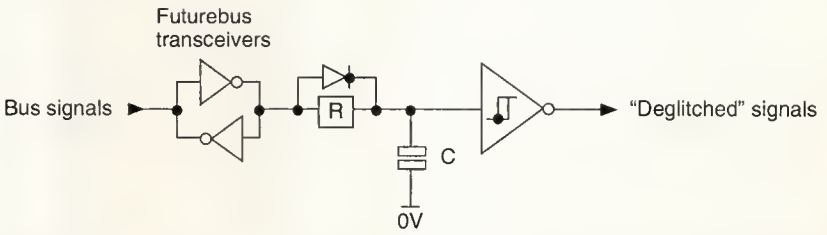


Figure 11. Glitch-filter circuit.

larly dk* and di*. Having said that, we point out that it is essential that no pair appears *simultaneously* released on the bus, lest a fast master overrun a slow slave.

We found this quite simple to guarantee, using the circuit of Figure 12 (implemented in the FSM PALs, of course). One signal controls a pair of gates that are cross-coupled to ensure that each gate's output can only be released when the other is asserted. This restriction imposes a one-gate-delay overlap at the changeover point, which is enough (in our case) to absorb all the skew in the bus drivers.

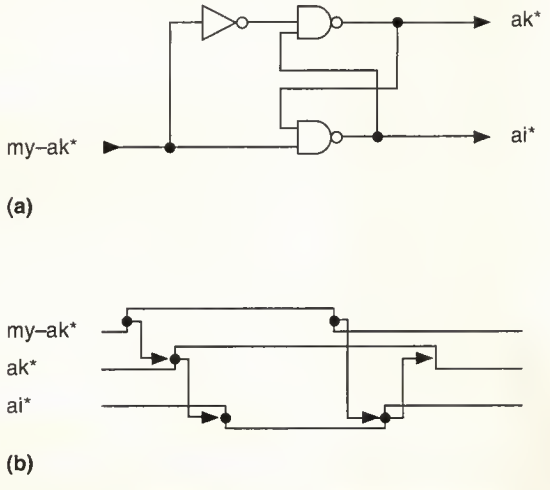


Figure 12. Complementary signal generation: circuit (a) and timing (b).

It would be nicer if this circuit were incorporated into the drivers themselves. Our one-gate-delay solution only really worked because National Semiconductor kindly supplied a worst-case intrapackage driver skew in response to our inquiry. The skew is considerably tighter than the interpackage skew. We then ensured that the complementary signals shared a bus-driver package. This data should be published as part of the driver specification.

Bus termination. We purchased bus termination units that contained a 2V reference source, to which each open-collector signal is connected via an appropriate pull-up resistor. When we first started to perform full-speed data transfer, we experienced very occasional parity errors. These errors never happened when we used the diagnostic interface to slow the system down. The problem turned out to be that the 2V reference was inadequately decoupled and was unable to provide enough current when all the data signals were driven low simultaneously. (With the usual inevitability of such things, the source dropped to exactly the 1.5V threshold, so our errors occurred only occasionally and were certainly dependent on the data.) The moral is to check your reference under dynamic conditions. This provision was so obvious that we only thought of it after trying everything else!

OUR FUTUREBUS INTERFACE consists of some 35 integrated circuits and occupies about 150 square centimeters. While our interface occupies only about 8 percent of the area of the boards, which are unusually large (37 cm × 52 cm), it would represent a substantial overhead on a smaller board format. Furthermore, our interface only implements a subset of the Futurebus protocols.

The largest system we have run so far contains 10 boards. We have not encountered any hardware problems attributable to the Futurebus or its interface on commissioned boards.

Once we became used to an asynchronous design approach, and had developed some CAD support for it, we found it easy to generate correct designs. We were apprehensive that our use of an unusual technique would lead to unforeseen problems, but in fact this was never the case.

Faults proved to be quite easy to track down, especially because they often cause the system to halt rather soon after the fault occurs. Thus the last few transitions captured by a logic analyzer are usually the ones of interest, in contrast to synchronous systems in which it is often difficult to capture the time interval of interest.

Asynchronous design is not well understood by engineers, or well supported by components and CAD tools. We think it deserves to be.

References

1. S.L. Peyton Jones et al., "GRIP—A High-Performance Architecture for Parallel Graph Reduction," *Proc. IFIP Conf. Functional Programming Languages and Computer Architecture*, G. Kahn, ed., Springer Verlag, Berlin, *LNCS (Lecture Notes in Computer Science) 274*, Sept. 1987, pp. 98-112.
2. R.V. Balakrishnan, "The Proposed IEEE 896 Futurebus—A Solution to the Bus-Driving Problem," *IEEE Micro*, Vol. 4, No. 4, Aug. 1984, pp. 23-27.
3. D.M. Taub, "Arbitration and Control Acquisition in the Proposed IEEE 896 Futurebus," *IEEE Micro*, Vol. 4, No. 4, Aug. 1984, pp. 28-41.
4. D.M. Taub, "Improved Control Acquisition Scheme for the IEEE

896 Futurebus," *IEEE Micro*, Vol. 7, No. 3, June 1987, pp. 52-62.
5. *Futurebus P896.1: A Backplane Bus Specification for Multiprocessor Architectures* (Draft 7.5a), Institute of Electrical and Electronics Engineers, Inc., New York, June 1987.
6. *Futurebus+ P896.1: Logical Layer Specifications* (Draft 8.2), IEEE, New York, Jan. 1990.
7. S.L. Peyton Jones, "Parallel Implementations of Functional Programming Languages," *Computer J.*, Vol. 32, No. 2, Apr. 1989, pp. 175-186.
8. W.I. Fletcher, *An Engineering Approach to Digital Design*, Prentice Hall, Englewood Cliffs, N.J., 1980.
9. C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, New York, 1980.
10. J.A. Brzozowski and J.C. Ebergen, "Recent Developments in the Design of Asynchronous Circuits," *Proc. Seventh Int'l Conf. Fundamentals of Computation Theory*, Aug. 1989, and Research Report CS-89-18, Computer Science Dept., Univ. of Waterloo, May 1989.
11. S.L. Peyton Jones, "A Practical Technique for Designing Asynchronous Finite-State Machines," Tech. Report, Dept. Computing Science, Univ. of Glasgow, Aug. 1989.
12. I.E. Sutherland, "Micropipelines," *Comm. ACM*, Vol. 32, No. 6, June 1989, pp. 720-738.

Simon L. Peyton Jones is a professor of computing science at the University of Glasgow, where he leads the Grasp project. Grasp centers on the compiler technology to support the new Haskell functional programming language, for the GRIP multiprocessor as well as uniprocessors. He also led the GRIP project and previously held a lecturing post at University College London for seven years. His main research interests are functional programming languages and their implementation.

Peyton Jones received a BA in electrical science and the Diploma in computer science, both from Trinity College, Cambridge. He authored a textbook about the implementation of functional languages using graph reduction. He is an affiliate member of the IEEE Computer Society and an associate member of the Association of Computing Machinery.

Mark S. Hardie currently free-lances as an electronic design engineer on a project for GEC Sensors, Debden, England. Previously, he worked on the hardware design for the GRIP project at UCL, where he had also worked on the design of a large VLSI-based RISC processor. He gained his BSc degree in electronics from Brighton.

Questions concerning this article may be addressed to Simon L. Peyton Jones, The University, Computing Science Department, Glasgow, G12 8QQ, Scotland; or Internet at simonpj@cs.glasgow.ac.uk.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate word on the Reader Service Card.

Low 174 Medium 175 High 176

FOR DISPLAY ADVERTISING INFORMATION, CONTACT:

Northern California and Pacific Northwest: John D. Vance & Associates, Inc., 4030 Moorpark Ave., Suite 116, San Jose, CA 95117; (408) 741-0354.

Southern California and Mountain States: Richard C. Faust Co., 24050 Madison St., Suite 101, Torrance, CA 90505; (213) 373-9604.

Midwest: The Kingwill Company, 4433 W. Touhy Ave., Suite 540, Lincolnwood, IL 60091; (708) 675-5755.

East Coast: Atlantic Representative Group, 349 Maple Place, Keyport, NJ 07735; (908) 739-1444.

New England: Arpin Associates, P.O. Box 6444, Holliston, MA 01746; (508) 429-8907.

Europe: Heinz J. Görgens, Parkstrasse 8a, D-4054 Nettetal 1 - Hinsbeck (F.R.G.); phone: (0 21 53) 8 99 88; fax: (0 21 53) 8 99 89.

Southwest/Southeast: Heidi Rex, Office, 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1264; (714) 821-8380.

For production information, conference, and classified advertising, contact Heidi Rex or Marian Tibayan.

IEEE MICRO, 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1264; phone (714) 821-8380; fax (714) 821-4010.

	RS #	Page #
Accel Technologies	83	60
Acumos	15	57
Ampro	14	57
AST Research	24	58
Chronology	80	60
Comco	31	59
Data Translation	19	58
Greenwich Instruments	18	58
IDE	29	59
Intel	25	59
Interaction Systems	28	59
LSI Logic	12	57
Mentor Graphics	85	60
Meta-Software	82	60
Micro Express	23	58
Morgan Kaufmann	1	4
Motorola	10,13,16,81	57,58,60
NMB Technologies	17	58
Planar	26	58
Radisys	22	58
Signal Analytics Corp.	21	58
Source III	86	60
STN International	—	C.IV
Telex Communications	27	59
Texas Instruments	11,20	57,58
Vantage Point Technologies	30	59
VLSI Technologies	84	60

Micro View

continued from p. 96

tions. New processors without architectural changes included:

- IDT's R3051/3052 and LSI Logic's LR33000, both using the standard MIPS architecture;
- LSI Logic's L64901, Cypress/Ross's 7C611, and Fujitsu's 86902, all minor modifications of existing Sparc implementations;
- AMD's 29050, a faster implementation of the 29000 with an on-chip floating-point unit;
- Intel's 960SA and 960SB, both cost-reduced versions of the 960KA and 960KB with a 16-bit bus;
- Intel's 960MM, marketed only to select military customers;
- Motorola's 68EC030, a 68030 without a memory management unit and with a lower price tag;
- Motorola's 68331 and 68340, both using the same 68020-like CPU32 core processor as originally introduced in the 68332; and
- National's 32CG160, which added on-chip peripherals to the 32CG16.

Several new processors extended existing architectures to improve performance:

- National's 32FX16 and 32GX320 include DSP extensions to the 32000 architecture and are designed especially for imaging applications.
- Fujitsu's 86930 Sparclite adds multiply, divide step, and table-lookup instructions to the standard Sparc architecture.
- Motorola's 68HC16 extends the popular 68HC11 microcontroller architecture to 16 bits.
- Motorola's 96002 DSP is similar to the 56001 but with a floating-point data path.
- AT&T's DSP3210 slightly extends the instruction set of the DSP32C while adding more memory and an interface to a host processor.
- TI's 340C40, a DSP with hardware support for multiprocessor networks, based on the 340C30 architecture, with minor additions.

Introductions of a few entirely new architectures for embedded control did occur last year. These architectures were designed to minimize transistor count and provide performance needed for specific applications. In most instances, the vendors supporting these architectures do not intend to compete in the broader embedded processor marketplace.

For example, Zilog licensed the Hyperstone architecture, but plans to use it primarily as a core processor in peripheral chips. Philips-Signetics introduced its very large instruction word (VLIW) Life architecture, but plans to offer it only as a custom product. Philips' engineers will optimize the architecture and write the software for each application.

Intel's Digital Video Interactive (DVI) processor and C&T's Programmable Universal Micro Accelerator (PUMA) processor are new architectures that primarily execute software provided by the chip vendor. In the case of the DVI chips, Intel supplies software for image compression and decompression, and the system designer can add software for image manipulation, special compression algorithms, or other functions. C&T's PUMA is a one-chip processor with an architecture similar to a typical 2900-family bit-slice system. The company intends PUMA to be used as a personal computer accelerator. C&T will supply microcode for Windows 3.0 and AutoCAD display and printer drivers.

Dedicated applications such as these are the best target for new architectures because the processor users don't need to develop any software and are thus insulated from the architecture.

Echelon's Neuron chips, which will be manufactured by Toshiba and Motorola, involve a new optimized architecture for a very specific class of applications: network nodes. These nodes control devices such as heaters, lights, and motors, or process signals from sensors. Echelon devised a unique architecture with three identical CPUs to handle the natural partitioning of the software into network media access control, higher level network protocols, and the user-specific application.

Two new, general-purpose embedded families recently arrived on the scene: NEC's K-series processors and Siemen's 80C166. Both companies have aimed these devices at perhaps the most ripe area for new architectures: embedded applications that need a reasonably low-cost processor but require more performance than an 8051 or 68HC11 provides.

What's next

In 1991, the pattern is likely to continue, with even fewer new architectures. The market is saturated, and the difficulties of introducing a new architecture are enormous. Modest performance gains are not enough to get the attention of designers already tired of evaluating new architectures.

Tremendous opportunities do exist for improving performance while reducing chip count and system complexity by adding more on-chip functions to existing architectures. This will be the focus of the most activity for the next few years. In addition, 64-bit architectures will emerge, but these primarily will be limited to high-end database servers and multiprocessor systems.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 201 Medium 202 High 203

Evolving architectures

Michael Slater

Microprocessor Report

(707) 823-4004

mslater@cup.portal.com

The 1980s brought forth a tremendous number of new architectures. Among complex instruction-set computing (CISC) architectures, considerable evolution occurred. The 8086 evolved to the 286 and then to the 386. The 286 and 386 were not simply new implementations of the 8086; they included significant architectural changes. The 68000 architecture—a better design from the start—experienced a more evolutionary change.

The introduction of reduced instruction-set computing (RISC) and RISC-like architectures was a major development. Notable architectures introduced included Fairchild's (now Intergraph's) Clipper, Sun's Sparc, Mips Computer Systems' R2000/R3000, Motorola's 88000, AMD's 29000, and Intel's 80960 and i860.

Several factors prompted this surge in new architectures. Most importantly, integrated circuit technology evolved to a point that made it possible to include hundreds of thousands of transistors on one chip. This development made one-chip, 32-bit microprocessors possible. Emergence of the RISC design approach into the commercial arena from projects at IBM, Stanford University, and University of California, Berkeley also contributed to this surge. Finally, commercial motives played a major part: Many semiconductor companies wanted to be in the microprocessor business, which meant they had to come up with a new architecture or find one to either license or otherwise copy.

Looking ahead

The 1990s will not see a repeat of this phenomenon, at least not in the first half of the decade. Instead, most new microprocessors will be new implementations of existing architectures, taking advantage of increasing transistor budgets to enhance performance, reduce system chip count, or both. The new microprocessors introduced in the past year provide clear evidence of this trend.

IBM's RS/6000 was the only new architecture introduced in 1990 for general-purpose systems. IBM is probably the only company in the world that could have successfully introduced a new architecture last year for this market. Its tremendous resources give IBM the luxury of time in establishing a new architecture. Also, IBM's marketing muscle attracts the attention of software developers and system buyers.

Even so, the RS/6000 would not have received much attention if it were not for its outstanding floating-point performance. Some of this performance comes from architectural features, such as the multiply-accumulate instruction. Additional performance comes from implementation characteristics, such as the capability to dispatch both a floating-point instruction and an integer instruction in the same clock cycle.

New developments

Other than the RS/6000, all the new processors introduced this past year for general-purpose systems are new implementations of existing architectures. Motorola's 68040 adds an on-chip floating-point unit and provides faster execution. However, it implements essentially the same architecture as the 68030. AMD's 286ZX is identical to the 286 except for its on-chip peripheral functions. Intel's 386SL is a 386-architecture processor with minor extensions (namely a new interrupt input with a private memory space) for power management.

New implementations of existing architectures also dominated embedded processor introduc-

continued on p. 95

0272-1732/91/0200-0096$01.00 © 1991 IEEE

1951-1991
40 YEARS OF SERVICE

IEEE COMPUTER SOCIETY
A member society of the
Institute of Electrical and Electronics Engineers, Inc.

THE FOLLOWING INFORMATION IS AVAILABLE:

Contact the Publications Office;
to facilitate handling, please request by number.

- Membership application, student #203, others #202
- Periodicals subscription form for individuals #200
- Periodicals subscription form for organizations #199
- Publications catalog #201
- Compmail electronic mail brochure #194
- Technical committee list/application #197
- Chapters lists, start-up procedures #193
- Student scholarship information #192
- Volunteer leaders/staff directory #196
- IEEE senior member grade application #204
  (requires ten years practice and significant performance in five of those ten)

To check membership status or report a change of address, call the IEEE toll-free number, 1-800-678-4333. Direct all other Computer Society related questions to the Publications Office.

PURPOSE

The IEEE Computer Society advances the theory and practice of computer science and engineering, promotes the exchange of technical information among 100,000 members worldwide, and provides a wide range of services to members and nonmembers.

MEMBERSHIP

Members receive the acclaimed monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

PUBLICATIONS AND ACTIVITIES

Computer. An authoritative, easy-to-read magazine containing tutorial and in-depth articles on topics across the computer field, plus news, conferences, calendar, interviews, and product reviews.

Periodicals. The society publishes six magazines and five research transactions. Refer to membership application or request information as noted above.

Conference Proceedings, Tutorial Texts, Standard Documents. The Computer Society Press publishes more than 100 titles every year.

Standards Working Groups. Over 100 of these groups produce IEEE standards used throughout the industrial world.

Technical Committees. More than 30 TCs publish newsletters, provide interaction with peers in specialty areas, and directly influence standards, conferences, and education.

Conferences/Education. The society holds about 100 conferences each year and sponsors many educational activities, including computing science accreditation.

Chapters. Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

OMBUDSMAN

Members experiencing problems — magazine delivery, membership status, or unresolved complaints — may write to the ombudsman at the Publications Office.

EXECUTIVE COMMITTEE

President: Duncan H. Lawrie*
University of Illinois
Dept. of Computer Science
1304 W. Springfield
Urbana, IL. 61801
(217) 333-3373

President-Elect: Bruce D. Shriver*
Past President: Helen M. Wood*

VP, Standards: Paul L. Borrill (1st VP)*
VP, Press Activities: Barry W. Johnson (2nd VP)*
VP, Conferences and Tutorials: Laurel V. Kaleda†
VP, Education: Raymond E. Miller†
VP, Membership Activities Ronald D. Williams†
VP, Publications: Ronald G. Hoelzeman†
VP, Technical Activities: Mario R. Barbacci*

Secretary: James H. Aylor*
Treasurer: Joseph Boykin†
Division V Director: Edward A. Parrish, Jr.†
Division VIII Director: Helen M. Wood*
Executive Director: T. Michael Elliott†

*voting member of the Board of Governors
†nonvoting member of the Board of Governors

BOARD OF GOVERNORS

Term Expiring 1991:
P. Bruce Berra, Michael Evangelist,
Ted Lewis, Raymond E. Miller, Earl E. Swartzlander, Jr.,
Joseph E. Urban, Thomas W. Williams

Term Expiring 1992:
James H. Aylor, Alicja I. Ellis, Tadao Ichikawa,
C.V. Ramamoorthy, Sallie V. Sheppard,
Harold Stone, Akihiko Yamada

Term Expiring 1993:
Fiorenza Albert-Howard, Jon T. Butler, Michael C. Mulder,
Yale N. Patt, Anneliese von Mayrhauser,
Benjamin W. Wah, Ronald Waxman

Next Board Meeting
March 1, 1991, 8:30 a.m.
Cathedral Hill Hotel, San Francisco, CA

SENIOR STAFF

Executive Director: T. Michael Elliott
Publisher: H. True Seaborn
Director, Conferences and Tutorials: Anne Marie Kelly
Director, Finance and Administration: Tod S. Heisler
Director, Board and Administrative Services: Violet S. Doan

COMPUTER SOCIETY OFFICES

Headquarters Office
1730 Massachusetts Ave. NW
Washington, DC 20036-1903
Phone (202) 371-0101
Fax: (202)728-9614

Publications Office
10662 Los Vaqueros Cir.
PO Box 3014
Los Alamitos, CA 90720-1264
Membership and General Information:
(714) 821-8380
Publication Orders: (800) 272-6657
Fax: (714) 821-4010

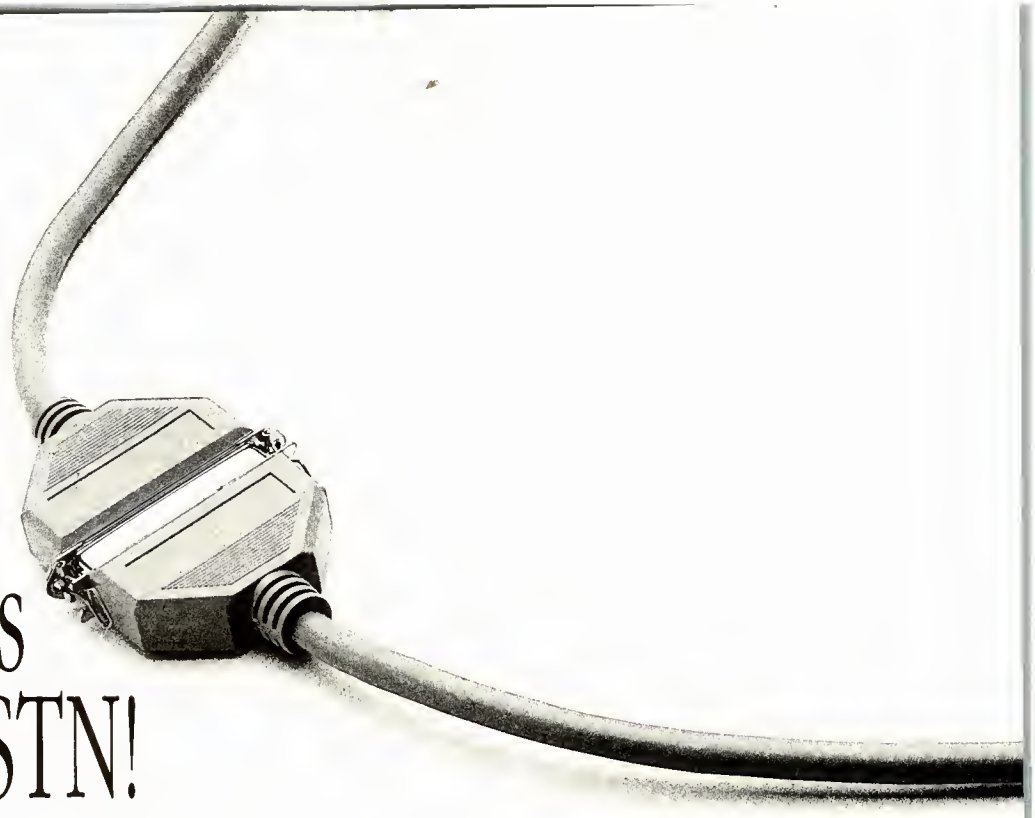
European Office
13, Ave. de L'Aquilon
B-1200 Brussels, Belgium
Phone: 32 (2) 770-21-98
Fax: 32 (2) 770-85-05

Asian Office
Ooshima Building
2-19-1 Minami-Aoyama, Minato-ku
Tokyo 107, Japan
Phone: 81 (3) 408-3118
Fax: 81 (3) 408-3553

IEEE OFFICERS

President: Eric E. Sumner
President Elect: Merrill W. Buckley, Jr.
Past President: Carleton A. Bayless
Secretary: Hugh Rudnick
Treasurer: Theodore W. Hissey, Jr

VP, Educational Activities: Richard S. Nichols
VP, Professional Activities: Michael J. Whitelaw
VP, Publication Activities: J. Thomas Cain
VP, Regional Activities: Robert T. H. Alden
VP, Technical Activities: Fernando Aldana



Make Great Connections Online On STN!

On STN International®, you can access databases covering every area of engineering. You'll find bibliographic and numeric files produced by leading scientific organizations, like AIChE, Engineering Information, IEEE, National Institute of Standards and Technology, and many others.

And everything about STN has been created to assist you in obtaining this information efficiently and economically. On STN, you'll find special features which enhance your searching, whether you're a novice or an expert.

One Command Language — Using a few simple commands, you'll be able to obtain information in more than 100 databases on STN. And STN's software, Messenger®, enables you to carry a search created in one database over to another on STN for further information.

Element Term Index — Through STN's Element Term (ET) Field, you can search for chemical symbols and other specialized notations. Using the ET Field can increase your accuracy AND efficiency.

Numeric Searching — On STN, you can search numeric values to locate substances having the property values you specify; search numeric ranges to find data you might otherwise miss; choose from SI, metric, engineering, or other units to display property values; search with substance names, names of properties, or CAS Registry Numbers® to retrieve numeric data.

As an STN customer, you can receive help from workshops, tutorial diskettes, STN Express® software, toll-free Help Desk, newsletters, and online document ordering. No one supports you like STN!

Make great connections on STN by filling out and returning the coupon below.

☐ **YES! Please tell me how to become a user of STN International.**

Name

Title

Organization

Address

City, State ZIP

Phone

MAIL TO: STN International, c/o Chemical Abstracts Service, Marketing Dept. 30091, P.O. Box 3012, Columbus, OH 43210

FAX TO: 1-614-447-3713

STN®
INTERNATIONAL
The Scientific & Technical Information Network